Background

You have a problem you wish to solve faster. What do you do?

Background

You have a problem you wish to solve faster. What do you do?

1. You think hard and devise a better solution

Background

You have a problem you wish to solve faster. What do you do?

1. You think hard and devise a better solution

Clearly this is a stupid thing to do. Programmers are much too lazy to do this

Background

You have a problem you wish to solve faster. What do you do?

1. You think hard and devise a better solution

Clearly this is a stupid thing to do. Programmers are much too lazy to do this

2. You get a faster processor

Background

You have a problem you wish to solve faster. What do you do?

1. You think hard and devise a better solution

Clearly this is a stupid thing to do. Programmers are much too lazy to do this

2. You get a faster processor

Better. This used to work, but not any more: processors have pretty much levelled off at around the 3-5GHz mark and don't seem to be getting faster

Background

3. You get a multicore machine and run the problem in parallel

Background

3. You get a multicore machine and run the problem in parallel

This *must* be the solution!

Background

3. You get a multicore machine and run the problem in parallel

This *must* be the solution!

Isn't it?

Background

3. You get a multicore machine and run the problem in parallel

This *must* be the solution!

Isn't it?

One purpose of this Unit is to make you realise this is actually the *hardest* way of doing it!

Background

3. You get a multicore machine and run the problem in parallel

This *must* be the solution!

Isn’t it?

One purpose of this Unit is to make you realise this is actually the *hardest* way of doing it!

In reality, No. 1 is best, then No. 2, lastly No. 3

Background

Consider the following:

Background

Consider the following:

- it takes one person ten months to build one house

Background

Consider the following:

- it takes one person ten months to build one house
- it takes ten people one month to build one house

Background

Consider the following:

- it takes one person ten months to build one house
- it takes ten people one month to build one house
- it takes 100 people one-tenth of a month to build one house

Background

Consider the following:

- it takes one person ten months to build one house
- it takes ten people one month to build one house
- it takes 100 people one-tenth of a month to build one house

Why is the last so implausible?

Background

When there are 100 people running about they will get in each others' way; fight over limited resources like bricks; some will have to sit and twiddle their thumbs while they wait for others to finish: you can't plumb a bathroom until the bathroom has been built

Background

When there are 100 people running about they will get in each others' way; fight over limited resources like bricks; some will have to sit and twiddle their thumbs while they wait for others to finish: you can't plumb a bathroom until the bathroom has been built

And so on

Background

When there are 100 people running about they will get in each others' way; fight over limited resources like bricks; some will have to sit and twiddle their thumbs while they wait for others to finish: you can't plumb a bathroom until the bathroom has been built

And so on

And when there are more workers, you will need more managers — not building themselves but making sure workers are doing the right things

Background

When there are 100 people running about they will get in each others’ way; fight over limited resources like bricks; some will have to sit and twiddle their thumbs while they wait for others to finish: you can’t plumb a bathroom until the bathroom has been built

And so on

And when there are more workers, you will need more managers — not building themselves but making sure workers are doing the right things

Simply adding more people won’t necessarily get it done faster

Background

When there are 100 people running about they will get in each others' way; fight over limited resources like bricks; some will have to sit and twiddle their thumbs while they wait for others to finish: you can't plumb a bathroom until the bathroom has been built

And so on

And when there are more workers, you will need more managers — not building themselves but making sure workers are doing the right things

Simply adding more people won't necessarily get it done faster

Sometimes adding more people will make it go *slower* as they get in each others' way

Background

But we can scale in a different way:

Background

But we can scale in a different way:

- it takes one person ten months to build one house

Background

But we can scale in a different way:

- it takes one person ten months to build one house
- it takes ten people ten months to build ten houses

Background

But we can scale in a different way:

- it takes one person ten months to build one house
- it takes ten people ten months to build ten houses
- it takes one person 100 months to build ten houses

Background

But we can scale in a different way:

- it takes one person ten months to build one house
- it takes ten people ten months to build ten houses
- it takes one person 100 months to build ten houses
- it takes ten people 100 months to build 100 houses

Background

But we can scale in a different way:

- it takes one person ten months to build one house
- it takes ten people ten months to build ten houses
- it takes one person 100 months to build ten houses
- it takes ten people 100 months to build 100 houses

This is much more believable: adding more people we can build more houses simultaneously

Background

But we can scale in a different way:

- it takes one person ten months to build one house
- it takes ten people ten months to build ten houses
- it takes one person 100 months to build ten houses
- it takes ten people 100 months to build 100 houses

This is much more believable: adding more people we can build more houses simultaneously

In reality, we won't get a perfect speedup like this, due to resource contention issues, but we can get pretty close

Background

Most people think parallel computing is about making things go *faster*

Background

Most people think parallel computing is about making things go *faster*

Up to a point, but they will soon be disappointed

Background

Most people think parallel computing is about making things go *faster*

Up to a point, but they will soon be disappointed

Much more likely to succeed is to make things *larger*

Background

Most people think parallel computing is about making things go *faster*

Up to a point, but they will soon be disappointed

Much more likely to succeed is to make things *larger*

This scales much better

Background

The first is *process* parallelism, also called *task* parallelism

Background

The first is *process* parallelism, also called *task* parallelism

The second is *data parallelism*

Background

The first is *process* parallelism, also called *task* parallelism

The second is *data parallelism*

Two very different ways of getting more in a given amount of time

Background

You all have had the situation where someone tries to help you do something and it's ended up taking *longer*

Background

You all have had the situation where someone tries to help you do something and it's ended up taking *longer*

There is the basic time it takes to solve the problem: then there are substantial overheads in the coordination of the parts of the solution

Background

You all have had the situation where someone tries to help you do something and it's ended up taking *longer*

There is the basic time it takes to solve the problem: then there are substantial overheads in the coordination of the parts of the solution

The overheads can easily be larger than the problem itself

Background

You all have had the situation where someone tries to help you do something and it's ended up taking *longer*

There is the basic time it takes to solve the problem: then there are substantial overheads in the coordination of the parts of the solution

The overheads can easily be larger than the problem itself

This is the reality of parallel computing

Background

You all have had the situation where someone tries to help you do something and it's ended up taking *longer*

There is the basic time it takes to solve the problem: then there are substantial overheads in the coordination of the parts of the solution

The overheads can easily be larger than the problem itself

This is the reality of parallel computing

Often a parallel version of a small problem will be *slower* than the sequential version

Background

You all have had the situation where someone tries to help you do something and it's ended up taking *longer*

There is the basic time it takes to solve the problem: then there are substantial overheads in the coordination of the parts of the solution

The overheads can easily be larger than the problem itself

This is the reality of parallel computing

Often a parallel version of a small problem will be *slower* than the sequential version

Only when the problem is made large enough to overcome the overheads will it become faster than doing it sequentially

Background

So cost (the number of cpu cycles) of a parallel computation
$=$ cost of computation $+$ cost of management of parallelism

Background

So cost (the number of cpu cycles) of a parallel computation $=$ cost of computation $+$ cost of management of parallelism

Ideally, we want the cost of management of parallelism to be minimal

Background

So cost (the number of cpu cycles) of a parallel computation $=$ cost of computation $+$ cost of management of parallelism

Ideally, we want the cost of management of parallelism to be minimal

But, if you are not careful, or the problem is such that this is inevitable, we can find that the cost of management of parallelism can dominate

Background

Another huge issue is that people have enough difficulties with programming sequential machines

Background

Another huge issue is that people have enough difficulties with programming sequential machines

Some would say that sequential programming is not yet a “solved” problem

Background

Another huge issue is that people have enough difficulties with programming sequential machines

Some would say that sequential programming is not yet a “solved” problem

Parallel programming is *much* harder

Background

Another huge issue is that people have enough difficulties with programming sequential machines

Some would say that sequential programming is not yet a "solved" problem

Parallel programming is *much* harder

If you think you understand parallel programing, you definitely don't

Background

You have all the issues of sequential programs **plus** lots more

Background

You have all the issues of sequential programs **plus** lots more

And they are issues that many programmers have difficulty even understanding

Background

You have all the issues of sequential programs **plus** lots more

And they are issues that many programmers have difficulty even understanding

Particularly as they have been trained to program for sequential machines and have habits and assumptions that are simply invalid for parallel machines

Background

Have I convinced you that parallel programming is difficult yet?

Background

Have I convinced you that parallel programming is difficult yet?

Well, it’s worse than you can imagine!

Background

You will see the terms *parallel* and *concurrent*, with some people using them interchangeably

Background

You will see the terms *parallel* and *concurrent*, with some people using them interchangeably

But it is sometimes important to make a distinction between the two

Background

You will see the terms *parallel* and *concurrent*, with some people using them interchangeably

But it is sometimes important to make a distinction between the two

concurrent means your computation has separately executable parts

Background

You will see the terms *parallel* and *concurrent*, with some people using them interchangeably

But it is sometimes important to make a distinction between the two

concurrent means your computation has separately executable parts

parallel means those parts are being executed *at the same time*

Background

You will see the terms *parallel* and *concurrent*, with some people using them interchangeably

But it is sometimes important to make a distinction between the two

concurrent means your computation has separately executable parts

parallel means those parts are being executed *at the same time*

Concurrency is about structure, parallelism is about execution

Background

So, “concurrent” means in parts, and those parts may or may not be running simultaneously

Background

So, “concurrent” means in parts, and those parts may or may not be running simultaneously

For example, they might be scheduled one at a time on a single core CPU)

Background

So, “concurrent” means in parts, and those parts may or may not be running simultaneously

For example, they might be scheduled one at a time on a single core CPU)

And “parallel” when we are explicitly talking about stuff running at the same time on multiple pieces of hardware

Background

So, “concurrent” means in parts, and those parts may or may not be running simultaneously

For example, they might be scheduled one at a time on a single core CPU)

And “parallel” when we are explicitly talking about stuff running at the same time on multiple pieces of hardware

> *Concurrency is about dealing with lots of things at once. Parallelism is about doing lots of things at once.*
> Rob Pike

Background

Asynchronous programming is an example of non-parallel concurrency.

Background

Asynchronous programming is an example of non-parallel concurrency.

This has been around for a long time in many disguises: *futures*, *promises*, *coroutines*, *generators* and others

Background

Asynchronous programming is an example of non-parallel concurrency.

This has been around for a long time in many disguises: *futures*, *promises*, *coroutines*, *generators* and others

The idea here is that when some code would block, e.g., waiting for some I/O, rather than the processor sitting and waiting doing nothing, the code should direct the processor to execute some other task

Background

Asynchronous programming is an example of non-parallel concurrency.

This has been around for a long time in many disguises: *futures*, *promises*, *coroutines*, *generators* and others

The idea here is that when some code would block, e.g., waiting for some I/O, rather than the processor sitting and waiting doing nothing, the code should direct the processor to execute some other task

Later, when the I/O is ready, the processor can come back to where it was and continue from there

Background

The code makes its own decision on what to do: moving between different parts of code, ensuring the processor is always actively working

Background

The code makes its own decision on what to do: moving between different parts of code, ensuring the processor is always actively working

This is scheduling *within* the code, without involvement of the Operating System

Background

The code makes its own decision on what to do: moving between different parts of code, ensuring the processor is always actively working

This is scheduling *within* the code, without involvement of the Operating System

As we know, any call to the OS entails a large amount of CPU overhead, which we avoid here

Background

The code makes its own decision on what to do: moving between different parts of code, ensuring the processor is always actively working

This is scheduling *within* the code, without involvement of the Operating System

As we know, any call to the OS entails a large amount of CPU overhead, which we avoid here

These are major points of async programming: avoid OS overheads and keep the processor busy

Background

So async code is concurrent (structural), but not parallel (execution)

Background

So async code is concurrent (structural), but not parallel (execution)

Programming async code is very complicated and shares many features with programming parallel code

Background

So async code is concurrent (structural), but not parallel (execution)

Programming async code is very complicated and shares many features with programming parallel code

Modern programming languages are starting to support async programming natively, e.g., JavaScript, Swift, C++, Rust, Python and more

Background

So async code is concurrent (structural), but not parallel (execution)

Programming async code is very complicated and shares many features with programming parallel code

Modern programming languages are starting to support async programming natively, e.g., JavaScript, Swift, C++, Rust, Python and more

Constructs in the languages hide varying amounts of the gory details of choosing and switching between tasks

Background

Async programming is good in cases where we have lots of tasks that mostly wait, e.g., I/O

Background

Async programming is good in cases where we have lots of tasks that mostly wait, e.g., I/O

Parallel programming is good in cases where we have lots of tasks that mostly compute

Background

Async programming is good in cases where we have lots of tasks that mostly wait, e.g., I/O

Parallel programming is good in cases where we have lots of tasks that mostly compute

Async is cooperative while parallel is preemptive

Background

Async programming is good in cases where we have lots of tasks that mostly wait, e.g., I/O

Parallel programming is good in cases where we have lots of tasks that mostly compute

Async is cooperative while parallel is preemptive

Async is for *waiting* in parallel

Background

In this unit we shall be concentrating on parallelism (though lots of what we say also applies to async programming, too)

Background

In this unit we shall be concentrating on parallelism (though lots of what we say also applies to async programming, too)

Exercise Reflect on how you might use **both** async and parallel programming in one program

Background

In contrast to concurrent and parallel, you might hear of *serial* and *sequential* both being used to describe non-concurrent/non-parallel systems

Background

In contrast to concurrent and parallel, you might hear of *serial* and *sequential* both being used to describe non-concurrent/non-parallel systems

Serial and sequential mean the same thing

Background

Moore's Law

Why is parallelism an important topic these days?

Background

Moore's Law

Why is parallelism an important topic these days?

There is a famous "law" that describes how hardware has progressed over the years

Background

Moore's Law

Why is parallelism an important topic these days?

There is a famous "law" that describes how hardware has progressed over the years

It is an **observation** on how the components in integrated circuits were shrinking over time as engineering advances were made:

Background

Moore's Law

Why is parallelism an important topic these days?

There is a famous "law" that describes how hardware has progressed over the years

It is an **observation** on how the components in integrated circuits were shrinking over time as engineering advances were made:

Moore's Law (1965):

> *the number of transistors in a chip doubles every two years*

Background

Moore's Law

There are a number of points to be made

Background

Moore's Law

There are a number of points to be made

- it's not a “law” in any real sense, but an *observation* on how chips progress

Background

Moore's Law

There are a number of points to be made

- it's not a "law" in any real sense, but an *observation* on how chips progress
- Moore did not say *speed* doubles, as often mis-quoted

Background

Moore's Law

There are a number of points to be made

- it's not a "law" in any real sense, but an *observation* on how chips progress
- Moore did not say *speed* doubles, as often mis-quoted
- some variants say "18 months" instead of "two years", but the history of this statement is complex

Background

Moore's Law

There are a number of points to be made

- it's not a "law" in any real sense, but an *observation* on how chips progress
- Moore did not say *speed* doubles, as often mis-quoted
- some variants say "18 months" instead of "two years", but the history of this statement is complex
- it is somewhat self-fulfilling, as engineers tend to use it as a target for the development of each next generation of chips

Background

Moore's Law

There is some economics in there, too: the profit margins on silicon wafers mean that it is better to have fewer larger chips than lots of smaller chips

Background

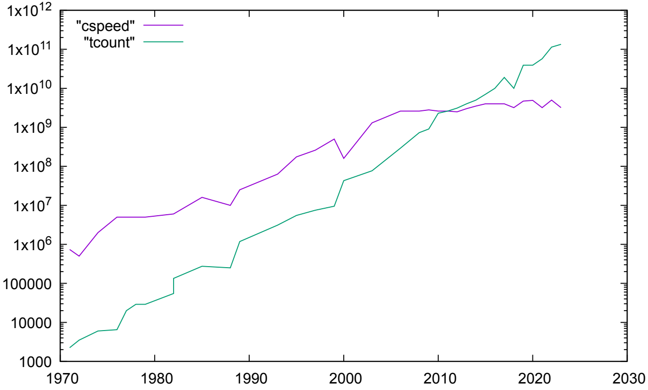
Moore's Law

There is some economics in there, too: the profit margins on silicon wafers mean that it is better to have fewer larger chips than lots of smaller chips

So CPUs tend to keep to the same area, meaning a CPU will have more and more transistors, not that we have more smaller CPUs

Background

Moore's Law



Log of speed and transistor count against date of Intel processors

Background

Moore's Law

We can see why people thought that Moore's Law was about speed: for a long time both transistor count and speed went up exponentially

Background

Moore's Law

We can see why people thought that Moore's Law was about speed: for a long time both transistor count and speed went up exponentially

In 2005 people would have said that CPUs would be running at 480GHz by 2020

Background

Moore's Law

We can see why people thought that Moore's Law was about speed: for a long time both transistor count and speed went up exponentially

In 2005 people would have said that CPUs would be running at 480GHz by 2020

However, over the last few years speed has stopped increasing

Background

Moore's Law

We can see why people thought that Moore's Law was about speed: for a long time both transistor count and speed went up exponentially

In 2005 people would have said that CPUs would be running at 480GHz by 2020

However, over the last few years speed has stopped increasing

But, crucially, the transistor count continues to increase

Background

Moore's Law

We can see why people thought that Moore's Law was about speed: for a long time both transistor count and speed went up exponentially

In 2005 people would have said that CPUs would be running at 480GHz by 2020

However, over the last few years speed has stopped increasing

But, crucially, the transistor count continues to increase

CPUs stay the same physical size

Background

Moore's Law

Engineer:
What are we going to do with those extra transistors?

Background

Moore's Law

Engineer:
What are we going to do with those extra transistors?

Marketer:
How are we going to convince people to buy the new CPUs?

Background

Moore's Law

Engineer:
What are we going to do with those extra transistors?

Marketer:
How are we going to convince people to buy the new CPUs?

Solution:
multicore processors

Background

Moore's Law

Engineer:
What are we going to do with those extra transistors?

Marketer:
How are we going to convince people to buy the new CPUs?

Solution:
multicore processors

Chips with more than one CPU on them

Background

So now chips in new PCs are all multicore

Background

So now chips in new PCs are all multicore

Dual and quad core is everywhere; 64 core processors are around; 128 cores are arriving soon (PC-style architecture)

Background

So now chips in new PCs are all multicore

Dual and quad core is everywhere; 64 core processors are around; 128 cores are arriving soon (PC-style architecture)

Many cores is great, but we are going to have to find out how to make best use of them

Background

So now chips in new PCs are all multicore

Dual and quad core is everywhere; 64 core processors are around; 128 cores are arriving soon (PC-style architecture)

Many cores is great, but we are going to have to find out how to make best use of them

But simply having two CPUs generally won't make our program go twice as fast: overheads like interference and communication between parts of the computation is going to be a problem

Background

To repeat: all this hardware is all wonderful except for one point

Background

To repeat: all this hardware is all wonderful except for one point

> *This computational power is only useful if we can write the software to exploit it*

Background

To repeat: all this hardware is all wonderful except for one point

> *This computational power is only useful if we can write the software to exploit it*

Your phone might have eight cores, but it is likely very little software it runs is capable of using all their power simultaneously

Background

To repeat: all this hardware is all wonderful except for one point

> *This computational power is only useful if we can write the software to exploit it*

Your phone might have eight cores, but it is likely very little software it runs is capable of using all their power simultaneously

Software is far behind hardware and has a lot to do to catch up

Background

To repeat: all this hardware is all wonderful except for one point

> *This computational power is only useful if we can write the software to exploit it*

Your phone might have eight cores, but it is likely very little software it runs is capable of using all their power simultaneously

Software is far behind hardware and has a lot to do to catch up

We are still in the dark regarding parallel software

Background

A Brief Aside

Note that Moore's Law also applies to memory: memory chips have been doubling in capacity at a similar (perhaps faster?) rate

Background

A Brief Aside

Note that Moore's Law also applies to memory: memory chips have been doubling in capacity at a similar (perhaps faster?) rate

But the *speed* of delivery of data from memory to processor(s) has always lagged behind the speed of processors

Background

A Brief Aside

Note that Moore's Law also applies to memory: memory chips have been doubling in capacity at a similar (perhaps faster?) rate

But the *speed* of delivery of data from memory to processor(s) has always lagged behind the speed of processors

Giving a problematic gap between speed of processors and speed of memory (both in bandwidth and latency)

Background

A Brief Aside

Note that Moore's Law also applies to memory: memory chips have been doubling in capacity at a similar (perhaps faster?) rate

But the *speed* of delivery of data from memory to processor(s) has always lagged behind the speed of processors

Giving a problematic gap between speed of processors and speed of memory (both in bandwidth and latency)

The gap has decreased a little over the last few years, but on the other hand multiple processors need more memory bandwidth

Background

A Brief Aside

Note that Moore's Law also applies to memory: memory chips have been doubling in capacity at a similar (perhaps faster?) rate

But the *speed* of delivery of data from memory to processor(s) has always lagged behind the speed of processors

Giving a problematic gap between speed of processors and speed of memory (both in bandwidth and latency)

The gap has decreased a little over the last few years, but on the other hand multiple processors need more memory bandwidth

We shall see memory is a big bottleneck in parallel systems

Background

Moore's Law

Moore's Law has been going for 58 years so far

Background

Moore's Law

Moore's Law has been going for 58 years so far

It must come to an end at some point: the end has been predicted many times in the past, but so far technology has kept moving onwards

Background

Moore's Law

Moore's Law has been going for 58 years so far

It must come to an end at some point: the end has been predicted many times in the past, but so far technology has kept moving onwards

Chip designers think it will keep going for several years yet, some predict decades

Background

Moore's Law

Moore's Law has been going for 58 years so far

It must come to an end at some point: the end has been predicted many times in the past, but so far technology has kept moving onwards

Chip designers think it will keep going for several years yet, some predict decades

Moore himself thinks perhaps it will last until 2025

Background

Moore's Law

Moore's Law has been going for 58 years so far

It must come to an end at some point: the end has been predicted many times in the past, but so far technology has kept moving onwards

Chip designers think it will keep going for several years yet, some predict decades

Moore himself thinks perhaps it will last until 2025

And — looking at Intel's products the last few years — it might currently be taking 5 years to double transistor counts

Background

Moore's Law

Exercise Some current top end chips have over 100 billion transistors, and 7000 cores. If Moore's Law continued, how many transistors and cores would they have in 10 years? In 20 years?

Exercise Read about Moore's Second Law (aka Rock's Law)

Background

Moore's Law

Software is getting slower more rapidly than hardware is becoming faster
Wirth's Law

Software efficiency halves every 18 months, compensating Moore's law
David May

The speed of software halves every 18 months
Gates' Law

What Intel giveth, Microsoft taketh away
Anon