Concurrency Control

Compiler Reordering

Thus, to be correct, the programmer needs to inform the compiler not to do these kinds of “optimisations”

Concurrency Control

Compiler Reordering

Thus, to be correct, the programmer needs to inform the compiler not to do these kinds of “optimisations”

Languages like C and Java have a `volatile` keyword:

```
volatile int cont;
```

Concurrency Control

Compiler Reordering

Thus, to be correct, the programmer needs to inform the compiler not to do these kinds of “optimisations”

Languages like C and Java have a `volatile` keyword:

```
volatile int cont;
```

tells the compiler not to mess around with such variables and assume that external operations might change their value

Concurrency Control

Compiler Reordering

But `volatile` was introduced for hardware/peripheral-related reasons and is *not* a way of fixing concurrency issues as they don't solve the whole problem, as the *hardware* needs telling, too

Concurrency Control

Compiler Reordering

But `volatile` was introduced for hardware/peripheral-related reasons and is *not* a way of fixing concurrency issues as they don't solve the whole problem, as the *hardware* needs telling, too

Summary: **don't** use `volatile` to try to solve parallelism problems, as is sometimes recommended

Concurrency Control

Hardware Reordering

Second problem: as it's not just the compiler that reorders things

Concurrency Control

Hardware Reordering

Second problem: as it's not just the compiler that reorders things

Modern CPUs use *out of order execution* on machine instructions to improve efficiency in superscalar architectures, where the *processor* can reorder instructions as it sees fit

Concurrency Control

Hardware Reordering

Second problem: as it's not just the compiler that reorders things

Modern CPUs use *out of order execution* on machine instructions to improve efficiency in superscalar architectures, where the *processor* can reorder instructions as it sees fit

For example, in the machine code for

Concurrency Control

Hardware Reordering

Second problem: as it's not just the compiler that reorders things

Modern CPUs use *out of order execution* on machine instructions to improve efficiency in superscalar architectures, where the *processor* can reorder instructions as it sees fit

For example, in the machine code for

```
x = y + z;
w = 2*u;
```

Concurrency Control

Hardware Reordering

Second problem: as it's not just the compiler that reorders things

Modern CPUs use *out of order execution* on machine instructions to improve efficiency in superscalar architectures, where the *processor* can reorder instructions as it sees fit

For example, in the machine code for

```
x = y + z;
w = 2*u;
```

Since loading from memory takes a long time the CPU might decide to start loading `u` *before* doing the sum

Concurrency Control

Hardware Reordering

Second problem: as it's not just the compiler that reorders things

Modern CPUs use *out of order execution* on machine instructions to improve efficiency in superscalar architectures, where the *processor* can reorder instructions as it sees fit

For example, in the machine code for

```
x = y + z;
w = 2*u;
```

Since loading from memory takes a long time the CPU might decide to start loading `u` *before* doing the sum

Again, this reduces the overall time the code takes to run as the multiply does not have to wait as long for `u` to arrive

Concurrency Control

Hardware Reordering

So, even given un-reordered code or machine code equivalent loading registers

`cont = 1;`	`load $r1, 1`
`x = 42;`	`load $r2, 42`

the CPU might *while running* decide the loads look independent and load `x` (`$r2`) first

Concurrency Control

Hardware Reordering

So, even given un-reordered code or machine code equivalent loading registers

```
cont = 1;                      load $r1, 1
x = 42;                        load $r2, 42
```

the CPU might *while running* decide the loads look independent and load `x` (`$r2`) first

Out of order execution is common in modern architectures

Concurrency Control

Thus we also need special code like

```
while (cont == 0) {/* nothing */}
memory_fence();
print x;
```

```
x = 42;
memory_fence();
cont = 1;
```

(details vary according to language and compiler) that tell the compiler *and* processor not to reorder things

Concurrency Control

Thus we also need special code like

```
while (cont == 0) {/* nothing */}
memory_fence();
print x;
```

```
x = 42;
memory_fence();
cont = 1;
```

(details vary according to language and compiler) that tell the compiler *and* processor not to reorder things

Firstly, the compiler will know not to try to move reads or writes across the call to `memory_fence()`

Concurrency Control

Thus we also need special code like

```
while (cont == 0) {/* nothing */}
memory_fence();
print x;
```

```
x = 42;
memory_fence();
cont = 1;
```

(details vary according to language and compiler) that tell the compiler *and* processor not to reorder things

Firstly, the compiler will know not to try to move reads or writes across the call to `memory_fence()`

Secondly, the `memory_fence()` would compile to a specific special machine instruction that tells the CPU's out of order mechanism not to move read or writes across this boundary

Concurrency Control

Thus we also need special code like

```
while (cont == 0) {/* nothing */}
memory_fence();
print x;
```

```
x = 42;
memory_fence();
cont = 1;
```

(details vary according to language and compiler) that tell the compiler *and* processor not to reorder things

Firstly, the compiler will know not to try to move reads or writes across the call to `memory_fence()`

Secondly, the `memory_fence()` would compile to a specific special machine instruction that tells the CPU's out of order mechanism not to move read or writes across this boundary

The first fence says not to read `x` too early, while the other says don't assign `cont` before `x`

Concurrency Control

Memory Consistency

In fact, in modern machine architectures you *must* use some primitive like a fence, or something that uses a fence (e.g., a semaphore), to ensure the intended behaviour

In fact, in modern machine architectures you *must* use some primitive like a fence, or something that uses a fence (e.g., a semaphore), to ensure the intended behaviour

Memory fences work (when you remember to use them) but prevent some correct optimisations. Thus more subtle mechanisms are also used

In fact, in modern machine architectures you *must* use some primitive like a fence, or something that uses a fence (e.g., a semaphore), to ensure the intended behaviour

Memory fences work (when you remember to use them) but prevent some correct optimisations. Thus more subtle mechanisms are also used

Exercise The above stops both reads and writes from being moved forward or back. Fences also come in variants that only block movement forward; or only movement back. Read about these

Concurrency Control

Third problem: other memory effects

Concurrency Control

Third problem: other memory effects

It is possible (in some machine architectures) for thread A to read the wrong value of `x`, *even if there is no out-of order execution*

Concurrency Control

Third problem: other memory effects

It is possible (in some machine architectures) for thread A to read the wrong value of `x`, *even if there is no out-of order execution*

It could be that B writes `x` and then writes `cont`; and A reads `cont` before reading `x`

Concurrency Control

Third problem: other memory effects

It is possible (in some machine architectures) for thread A to read the wrong value of `x`, *even if there is no out-of order execution*

It could be that B writes `x` and then writes `cont`; and A reads `cont` before reading `x`

But, due to caching (or other weirdness) it can be that B's write to `cont` reaches A before its write to `x`

Concurrency Control

Third problem: other memory effects

It is possible (in some machine architectures) for thread A to read the wrong value of `x`, *even if there is no out-of order execution*

It could be that B writes `x` and then writes `cont`; and A reads `cont` before reading `x`

But, due to caching (or other weirdness) it can be that B's write to `cont` reaches A before its write to `x`

So A reads the new value of `cont` but the old value of `x`, as its view of `x` has not yet been updated

Concurrency Control

Memory Consistency

The specification for a parallel language needs a *memory model* to describe how memory reads and writes are visible to multiple processors

Concurrency Control

Memory Consistency

The specification for a parallel language needs a *memory model* to describe how memory reads and writes are visible to multiple processors

This involves the use of special language constructs and special memory access operations to inform the compiler and hardware about what kinds of reordering are allowable and what kinds of *memory consistency* across processors are needed

Concurrency Control

Memory Consistency

And this is the problem: languages like C (and C++, and Java, and . . .) were conceived before memory models were necessary

Concurrency Control

Memory Consistency

And this is the problem: languages like C (and C++, and Java, and . . .) were conceived before memory models were necessary

So they didn't have them

Concurrency Control

Memory Consistency

And this is the problem: languages like C (and C++, and Java, and . . .) were conceived before memory models were necessary

So they didn't have them

Updates to the language standards are trying to retrofit memory models, but sometimes it's very difficult to fit new ideas into an old language

Concurrency Control

Memory Consistency

And this is the problem: languages like C (and C++, and Java, and ...) were conceived before memory models were necessary

So they didn't have them

Updates to the language standards are trying to retrofit memory models, but sometimes it's very difficult to fit new ideas into an old language

Further, programmers need to be (re)trained to understand these things

So, for example, the programmer may decide that some reads or some writes may be reordered, while others should not

Concurrency Control

Memory Consistency

So, for example, the programmer may decide that some reads or some writes may be reordered, while others should not

Generally, the programmer must understand the issues involved and use the right constructs in the right places

Concurrency Control

Memory Consistency

So, for example, the programmer may decide that some reads or some writes may be reordered, while others should not

Generally, the programmer must understand the issues involved and use the right constructs in the right places

Allowing just enough flexibility for the compiler/hardware to be efficient, while still correct code

Concurrency Control

Memory Consistency

So, for example, the programmer may decide that some reads or some writes may be reordered, while others should not

Generally, the programmer must understand the issues involved and use the right constructs in the right places

Allowing just enough flexibility for the compiler/hardware to be efficient, while still correct code

Thus allowing the system to reduce synchronisation and increase parallelism

Concurrency Control

Memory Consistency

Fortunately for us, if we use primitives (locks, semaphores, and so on) and higher-level constructs they will look after the details for us

Fortunately for us, if we use primitives (locks, semaphores, and so on) and higher-level constructs they will look after the details for us

As long as we use them!

Concurrency Control

Memory Consistency

Fortunately for us, if we use primitives (locks, semaphores, and so on) and higher-level constructs they will look after the details for us

As long as we use them!

So: if you have a cross-thread relationship, use a parallelism mechanism, don't just wing it

Exercise Read about memory consistency. Including: memory fences, *strict consistency*, *strong consistency*, *causal consistency*, *weak consistency*, *sequentially consistent*, *acquire-release*, *relaxed*, *consume*, etc.

Exercise Read about how modern C and C++ standards address the memory consistency issue

Exercise Read about the difference between Java's memory model and C/C++'s model (and what `volatile` does in each)

Exercise Read about the difference between the Intel (x86) memory model and the Arm memory model

Exercise And read about the memory problems that Apple's Arm M1 and later chips have in trying to support old x86 code via an instruction translator (Rosetta)

Concurrency Control

Threads

So now you have the tools to hand: thread creation to run things concurrently/in parallel, and primitives to control races

Concurrency Control

Threads

So now you have the tools to hand: thread creation to run things concurrently/in parallel, and primitives to control races

An important note on the cost of thread creation: they are not free!

Concurrency Control

Threads

So now you have the tools to hand: thread creation to run things concurrently/in parallel, and primitives to control races

An important note on the cost of thread creation: they are not free!

But, in a good OS implementation, they are relatively cheap

Concurrency Control

Threads

So now you have the tools to hand: thread creation to run things concurrently/in parallel, and primitives to control races

An important note on the cost of thread creation: they are not free!

But, in a good OS implementation, they are relatively cheap

Depending on the operating system, it can take hundreds or thousands of CPU instructions to create or destroy a thread

Concurrency Control

Threads

So now you have the tools to hand: thread creation to run things concurrently/in parallel, and primitives to control races

An important note on the cost of thread creation: they are not free!

But, in a good OS implementation, they are relatively cheap

Depending on the operating system, it can take hundreds or thousands of CPU instructions to create or destroy a thread

For the "hello" examples above it probably would not be worth creating new threads, but be faster to run the `printfs` sequentially

Concurrency Control

Threads

So now you have the tools to hand: thread creation to run things concurrently/in parallel, and primitives to control races

An important note on the cost of thread creation: they are not free!

But, in a good OS implementation, they are relatively cheap

Depending on the operating system, it can take hundreds or thousands of CPU instructions to create or destroy a thread

For the “hello” examples above it probably would not be worth creating new threads, but be faster to run the `printfs` sequentially

(But, remember, raw speed is not necessarily the target for parallelism)

Concurrency Control

Threads

A rough test on my PC indicates that the overhead of creating and joining one thread is about the same amount of time as doing 2000 floating point operations

Concurrency Control

Threads

A rough test on my PC indicates that the overhead of creating and joining one thread is about the same amount of time as doing 2000 floating point operations

Exercise That is for a particular OS and a particular CPU. Find out how long it takes to create a thread on your computer and OS

Concurrency Control

Threads

You have to judge whether it is worthwhile paying the creation overhead

Concurrency Control

Threads

You have to judge whether it is worthwhile paying the creation overhead

And there is the additional cost of *context switching* between threads when there are more threads than processors

Concurrency Control

Threads

You have to judge whether it is worthwhile paying the creation overhead

And there is the additional cost of *context switching* between threads when there are more threads than processors

The thread model of parallelism leads one to write programs with large numbers of threads

Concurrency Control

Threads

You have to judge whether it is worthwhile paying the creation overhead

And there is the additional cost of *context switching* between threads when there are more threads than processors

The thread model of parallelism leads one to write programs with large numbers of threads

Probably more than there are processors in the system, particularly when you take into account the threads in the other processes running in the system

Concurrency Control

Threads

This means that threads need to be scheduled, just like processes

Concurrency Control

Threads

This means that threads need to be scheduled, just like processes

And this has a cost, just like processes

Concurrency Control

Threads

This means that threads need to be scheduled, just like processes

And this has a cost, just like processes

It is easy to make so many threads that the OS starts thrashing

This means that threads need to be scheduled, just like processes

And this has a cost, just like processes

It is easy to make so many threads that the OS starts thrashing

You need to be careful about how many threads to create!

Concurrency Control

Threads

This means that threads need to be scheduled, just like processes

And this has a cost, just like processes

It is easy to make so many threads that the OS starts thrashing

You need to be careful about how many threads to create!

Typically, creating a (POSIX) thread when you need it, and then destroying it when done is costly and not a good approach

Concurrency Control

Threads

This means that threads need to be scheduled, just like processes

And this has a cost, just like processes

It is easy to make so many threads that the OS starts thrashing

You need to be careful about how many threads to create!

Typically, creating a (POSIX) thread when you need it, and then destroying it when done is costly and not a good approach

The objective is to give a thread as much computation as possible, perhaps repeated or multiple tasks

Trying to address the cost of thread creation and deletion leads some people to the *thread pool* model of parallelism

Concurrency Control

Thread Pools

Trying to address the cost of thread creation and deletion leads some people to the *thread pool* model of parallelism

Your program creates a pool of threads (not too many, not too few!) once and reuses them multiple times

Concurrency Control

Thread Pools

Trying to address the cost of thread creation and deletion leads some people to the *thread pool* model of parallelism

Your program creates a pool of threads (not too many, not too few!) once and reuses them multiple times

Each thread is given a task as is necessary; it does it and then goes back for another task

Concurrency Control

Thread Pools

You pay the cost of creation just once at the start (and destruction just once at the end), rather than once per thread use

Concurrency Control

Thread Pools

You pay the cost of creation just once at the start (and destruction just once at the end), rather than once per thread use

Though there is a cost in the pool task management mechanisms

Concurrency Control

Thread Pools

You pay the cost of creation just once at the start (and destruction just once at the end), rather than once per thread use

Though there is a cost in the pool task management mechanisms

But these threads have a long life, and do many things

Apple's *Grand Central Dispatch* (GCD) does thread pooling at a higher level: system-wide

Concurrency Control

Thread Pools

Apple's *Grand Central Dispatch* (GCD) does thread pooling at a higher level: system-wide

The OS manages threads across all processes running, not just within each process

Concurrency Control

Thread Pools

Apple's *Grand Central Dispatch* (GCD) does thread pooling at a higher level: system-wide

The OS manages threads across all processes running, not just within each process

More on GCD later (in particular, its costs), but note this is in contrast to the model of each *program* creating and destroying threads as it needs them, as we were doing previously

Concurrency Control

POSIX

As mentioned previously, POSIX pthreads are a very popular library-based mechanism to support parallelism (actually: concurrency)

Concurrency Control

POSIX

As mentioned previously, POSIX pthreads are a very popular library-based mechanism to support parallelism (actually: concurrency)

We have just scratched the surface of POSIX

Concurrency Control

POSIX

As mentioned previously, POSIX pthreads are a very popular library-based mechanism to support parallelism (actually: concurrency)

We have just scratched the surface of POSIX

There are lots of other functions described by the POSIX standard: try

```
man -k pthread
```

and

```
man 7 pthreads
```

on Linux for an overview

Concurrency Control

Non-POSIX

Windows has something similar to POSIX threads: different names for the functions, but similar enough to be confusing

Concurrency Control

Non-POSIX

Windows has something similar to POSIX threads: different names for the functions, but similar enough to be confusing

They do provide an implementation of POSIX threads, but MS would rather you use their own thread library: MS are not interested in portability across OSs

Concurrency Control

Non-POSIX

Windows has something similar to POSIX threads: different names for the functions, but similar enough to be confusing

They do provide an implementation of POSIX threads, but MS would rather you use their own thread library: MS are not interested in portability across OSs

Apple macOS, like Linux, has good POSIX coverage

Concurrency Control

Other Threads

It is worthwhile mentioning that there are many other kinds of threads, mostly invented to try to overcome the costs of (a) thread creation/deletion and (b) context switching between threads

Concurrency Control

Other Threads

It is worthwhile mentioning that there are many other kinds of threads, mostly invented to try to overcome the costs of (a) thread creation/deletion and (b) context switching between threads

They have names like *fibres*, *coroutines*, *protothreads*, *microthreads*, *light-weight processes* and so on

Concurrency Control

Other Threads

For example, some languages, e.g., Go (“goroutines”) and Erlang (“processes”), have very *lightweight threads* as part of the language

Concurrency Control

Other Threads

For example, some languages, e.g., Go (“goroutines”) and Erlang (“processes”), have very *lightweight threads* as part of the language

These are scheduled by the language runtime across system threads

Concurrency Control

Other Threads

For example, some languages, e.g., Go (“goroutines”) and Erlang (“processes”), have very *lightweight threads* as part of the language

These are scheduled by the language runtime across system threads

They are very cheap to create, and allow thousands or millions of “threads” to be active

Concurrency Control

Other Threads

For example, some languages, e.g., Go (“goroutines”) and Erlang (“processes”), have very *lightweight threads* as part of the language

These are scheduled by the language runtime across system threads

They are very cheap to create, and allow thousands or millions of “threads” to be active

They encourage the use of massive threading at the cost of overhead from a more complicated language runtime

Concurrency Control

Other Threads

For example, some languages, e.g., Go (“goroutines”) and Erlang (“processes”), have very *lightweight threads* as part of the language

These are scheduled by the language runtime across system threads

They are very cheap to create, and allow thousands or millions of “threads” to be active

They encourage the use of massive threading at the cost of overhead from a more complicated language runtime

More discussion of Go and Erlang later

More Libraries

We were discussing library-based parallelism

More Libraries

We were discussing library-based parallelism

Taking a sequential language and using a parallel library

More Libraries

We were discussing library-based parallelism

Taking a sequential language and using a parallel library

But this has the dangers of the sequential language not understanding parallelism and mis-optimising

More Libraries

We were discussing library-based parallelism

Taking a sequential language and using a parallel library

But this has the dangers of the sequential language not understanding parallelism and mis-optimising

But library-based parallelism is very popular: particularly if we avoid shared memory

More Libraries

Another important library-based solution is the *Message Passing Interface* (MPI) and we shall look at this later when we talk about distributed memory systems

More Libraries

Another important library-based solution is the *Message Passing Interface* (MPI) and we shall look at this later when we talk about distributed memory systems

We shall just note here that MPI is an example of one library-based technique that is quite popular: write code that is sequential, or modestly parallel, but call library functions that do what we want to achieve that are parallel—and written by somebody else

More Libraries

Another important library-based solution is the *Message Passing Interface* (MPI) and we shall look at this later when we talk about distributed memory systems

We shall just note here that MPI is an example of one library-based technique that is quite popular: write code that is sequential, or modestly parallel, but call library functions that do what we want to achieve that are parallel—and written by somebody else

Another example, the *Basic Linear Algebra Subprograms* (BLAS)

More Libraries

The BLAS are a (standard for a) collection of functions that implement various algorithms in linear algebra: vector sums; matrix multiplication; vector dot products; etc. for various representations of these datatypes

More Libraries

The BLAS are a (standard for a) collection of functions that implement various algorithms in linear algebra: vector sums; matrix multiplication; vector dot products; etc. for various representations of these datatypes

Implementations are written by people who really understand what they are doing in terms of making the best use of hardware: in particular parallel hardware

More Libraries

The BLAS are a (standard for a) collection of functions that implement various algorithms in linear algebra: vector sums; matrix multiplication; vector dot products; etc. for various representations of these datatypes

Implementations are written by people who really understand what they are doing in terms of making the best use of hardware: in particular parallel hardware

If you write your application to use the BLAS your code will be using this expertise

More Libraries

The BLAS are a (standard for a) collection of functions that implement various algorithms in linear algebra: vector sums; matrix multiplication; vector dot products; etc. for various representations of these datatypes

Implementations are written by people who really understand what they are doing in terms of making the best use of hardware: in particular parallel hardware

If you write your application to use the BLAS your code will be using this expertise

If someone comes out with an improved implementation of the BLAS that goes twice as fast, your code will automatically go twice as fast (in the BLAS bit)

More Libraries

They really can be a factor of two difference *on the same hardware*

More Libraries

They really can be a factor of two difference *on the same hardware*

BLAS libraries are typically tuned to the version of the processor in your machine, taking into account cache sizes; memory speeds and so on

More Libraries

They really can be a factor of two difference *on the same hardware*

BLAS libraries are typically tuned to the version of the processor in your machine, taking into account cache sizes; memory speeds and so on

The GotoBLAS, written by Goto, are recognised as being particularly good

More Libraries

They really can be a factor of two difference *on the same hardware*

BLAS libraries are typically tuned to the version of the processor in your machine, taking into account cache sizes; memory speeds and so on

The GotoBLAS, written by Goto, are recognised as being particularly good

His implementation contains chunks of processor-specific assembler and pays particular attention to the sizes of blocks of data, matching them carefully to cache sizes

More Libraries

Many other libraries exist: for example, the *template* approach

More Libraries

Many other libraries exist: for example, the *template* approach

This is a standard header file with a library of code behind it that introduces a bunch of new classes to aid parallel computation

More Libraries

Many other libraries exist: for example, the *template* approach

This is a standard header file with a library of code behind it that introduces a bunch of new classes to aid parallel computation

For example, C++ AMP (Accelerated Massive Parallelism) from Microsoft defines some parallel container types with methods that act concurrently on them

More Libraries

Many other libraries exist: for example, the *template* approach

This is a standard header file with a library of code behind it that introduces a bunch of new classes to aid parallel computation

For example, C++ AMP (Accelerated Massive Parallelism) from Microsoft defines some parallel container types with methods that act concurrently on them

E.g., `concurrency::parallel_for_each(...)`

More Libraries

Many other libraries exist: for example, the *template* approach

This is a standard header file with a library of code behind it that introduces a bunch of new classes to aid parallel computation

For example, C++ AMP (Accelerated Massive Parallelism) from Microsoft defines some parallel container types with methods that act concurrently on them

E.g., `concurrency::parallel_for_each(...)`

The details are hidden from the programmer, who gets a fairly simple API to work with

More Libraries

There are many other template libraries for C++ (a language very suited to this approach):

- Parallel Patterns Library (PPL) from Microsoft
- Thrust from Nividia
- Intel Threading Building Blocks (TBB)
- Boost
- Etc.

But you do need to be careful using them: they do make writing parallel code simpler, but they don't necessarily prevent you from using them incorrectly!