Parallel

Parallel computers are becoming ever more important

Parallel

Parallel computers are becoming ever more important

Most programming languages were designed decades ago in a uniprocessor world

Parallel

Parallel computers are becoming ever more important

Most programming languages were designed decades ago in a uniprocessor world

Some languages families are naturally parallel

- Declarative
- Functional

Parallel

Parallel computers are becoming ever more important

Most programming languages were designed decades ago in a uniprocessor world

Some languages families are naturally parallel

- Declarative
- Functional

As declarative languages don't specify how to do something, the system is free to do things in the most efficient way it can: and this includes in parallel

Parallel

The nature of functional languages is such that it is relatively easy to make parts run in parallel due to them having no global state, so no possibility of unexpected interference between different parts of the program

Parallel

The nature of functional languages is such that it is relatively easy to make parts run in parallel due to them having no global state, so no possibility of unexpected interference between different parts of the program

Unexpected interference is a major problem in parallel programs

Parallel

The nature of functional languages is such that it is relatively easy to make parts run in parallel due to them having no global state, so no possibility of unexpected interference between different parts of the program

Unexpected interference is a major problem in parallel programs

We all have had the experience of working on something with other people where someone else changes something while you are not looking, thereby messing up what you were trying to do

Parallel

The nature of functional languages is such that it is relatively easy to make parts run in parallel due to them having no global state, so no possibility of unexpected interference between different parts of the program

Unexpected interference is a major problem in parallel programs

We all have had the experience of working on something with other people where someone else changes something while you are not looking, thereby messing up what you were trying to do

Parallel programming is this a trillion times faster

Parallel

The nature of functional languages is such that it is relatively easy to make parts run in parallel due to them having no global state, so no possibility of unexpected interference between different parts of the program

Unexpected interference is a major problem in parallel programs

We all have had the experience of working on something with other people where someone else changes something while you are not looking, thereby messing up what you were trying to do

Parallel programming is this a trillion times faster

Plus a lot of other less obvious problems

Parallel

There have been many attempts to take existing sequential languages and tweak them to support parallelism

Parallel

There have been many attempts to take existing sequential languages and tweak them to support parallelism

The idea is to take advantage of the programmer's familiarity with the legacy sequential language

Parallel

There have been many attempts to take existing sequential languages and tweak them to support parallelism

The idea is to take advantage of the programmer's familiarity with the legacy sequential language

Which may lure them into a false sense that they understand what they are doing in the parallel version of the language

Parallel

There have been many attempts to take existing sequential languages and tweak them to support parallelism

The idea is to take advantage of the programmer's familiarity with the legacy sequential language

Which may lure them into a false sense that they understand what they are doing in the parallel version of the language

And the legacy language likely has no in-built prevention of things that are safe sequentially, but dangerous in parallel (e.g., updating a shared variable)

Parallel

Some languages were designed to be parallel from scratch

Parallel

Some languages were designed to be parallel from scratch

Occam (1993): derived from a mathematical notation for parallel processes. Was going to be big, but the hardware of the time couldn't cope

Parallel

Some languages were designed to be parallel from scratch

Occam (1993): derived from a mathematical notation for parallel processes. Was going to be big, but the hardware of the time couldn't cope

Strand: declarative, derived from Prolog

Parallel

Some languages were designed to be parallel from scratch

Occam (1993): derived from a mathematical notation for parallel processes. Was going to be big, but the hardware of the time couldn't cope

Strand: declarative, derived from Prolog

Erlang: functional. Used in real applications

Parallel

Some languages were designed to be parallel from scratch

Occam (1993): derived from a mathematical notation for parallel processes. Was going to be big, but the hardware of the time couldn't cope

Strand: declarative, derived from Prolog

Erlang: functional. Used in real applications

Go: intended to be a modern version of C, designed by the designers of C. Also called *Golang* to aid search on the Web

Parallel

Some languages were designed to be parallel from scratch

Occam (1993): derived from a mathematical notation for parallel processes. Was going to be big, but the hardware of the time couldn't cope

Strand: declarative, derived from Prolog

Erlang: functional. Used in real applications

Go: intended to be a modern version of C, designed by the designers of C. Also called *Golang* to aid search on the Web

Rust: designed to be a memory-safe replacement for C

Parallel

Feet

- Occam: You shoot both your feet with several guns at once
- Go: To shoot yourself in the foot you must first import the `unsafe` package
- Rust: you try to shoot yourself in the foot, but you can't as the gun has immutably borrowed your foot

Parallel

In practice, most parallel programmers use legacy languages such as C, C++ and Java

Parallel

In practice, most parallel programmers use legacy languages such as C, C++ and Java

With tweaks to the language, or by using parallel libraries of code

Parallel

In practice, most parallel programmers use legacy languages such as C, C++ and Java

With tweaks to the language, or by using parallel libraries of code

And they typically have a *lot* of parallelism (and other!) bugs

Parallel

But there are many large, successful parallel systems built using these languages

Parallel

But there are many large, successful parallel systems built using these languages

Even though it would be better to throw away all the code and start again with a language that was designed with parallelism in mind

Parallel

But there are many large, successful parallel systems built using these languages

Even though it would be better to throw away all the code and start again with a language that was designed with parallelism in mind

But the economics of doing this (time to train programmers, time to write and debug new code, etc.) means it rarely happens

Parallel

The conclusion is: if you are starting a new project that needs to be parallel, think carefully about the language you are going to use

Parallel

The conclusion is: if you are starting a new project that needs to be parallel, think carefully about the language you are going to use

If you are taking an existing project and trying to make it parallel: be *very* careful

Parallel

The conclusion is: if you are starting a new project that needs to be parallel, think carefully about the language you are going to use

If you are taking an existing project and trying to make it parallel: be *very* careful

There is much more to be said. There is a whole final-year Unit on parallelism!

Memory Management

Some languages manage memory allocation and deallocation for you, some don't

Memory Management

Some languages manage memory allocation and deallocation for you, some don’t

For example, some languages have garbage collection, while others require the programmer to take care over memory deallocation

Memory Management

In code

Memory Management

In code

```
bigclass x = new bigclass(1);
x = new bigclass(2);
```

Memory Management

In code

```
bigclass x = new bigclass(1);
x = new bigclass(2);
```

or

Memory Management

In code

```
bigclass x = new bigclass(1);
x = new bigclass(2);
```

or

```
(setq x (make <bigclass> 1))
(setq x (make <bigclass> 2))
```

Memory Management

In code

```
bigclass x = new bigclass(1);
x = new bigclass(2);
```

or

```
(setq x (make <bigclass> 1))
(setq x (make <bigclass> 2))
```

They both allocate memory to store an instance of `bigclass` (initialised with `1`); then they allocate more memory to store another instance of `bigclass` (initialised with `2`)

Memory Management

In code

```
bigclass x = new bigclass(1);
x = new bigclass(2);
```

or

```
(setq x (make <bigclass> 1))
(setq x (make <bigclass> 2))
```

They both allocate memory to store an instance of `bigclass` (initialised with `1`); then they allocate more memory to store another instance of `bigclass` (initialised with `2`)

But the memory allocated to the object 1 is no longer accessible to the program as the reference to the object stored in the variable `x` has been overwritten

Memory Management

GC

Being inaccessible to the program means that object can have no use to the program, it cannot affect the program, but just sits there occupying memory

Memory Management

GC

Being inaccessible to the program means that object can have no use to the program, it cannot affect the program, but just sits there occupying memory

But, equally, being inaccessible, that program couldn't tell if that memory was reused for something else

Memory Management

GC

Being inaccessible to the program means that object can have no use to the program, it cannot affect the program, but just sits there occupying memory

But, equally, being inaccessible, that program couldn't tell if that memory was reused for something else

Thus: this memory *can't* be accessed by the program, so it *can* be used for something else

Memory Management

GC

Being inaccessible to the program means that object can have no use to the program, it cannot affect the program, but just sits there occupying memory

But, equally, being inaccessible, that program couldn't tell if that memory was reused for something else

Thus: this memory *can't* be accessed by the program, so it *can* be used for something else

Otherwise, that memory is just garbage. We can use a *garbage collector* to search out such inaccessible memory and reclaim it and reuse it

Memory Management

GC

Note, if we do nothing, the program will likely eventually run out of memory

Memory Management

GC

Note, if we do nothing, the program will likely eventually run out of memory

A garbage collector is code, usually part of the language runtime, that periodically searches through memory for memory that is no longer accessible to the program and reclaims it and so allows it to be reused

Memory Management

GC

Note, if we do nothing, the program will likely eventually run out of memory

A garbage collector is code, usually part of the language runtime, that periodically searches through memory for memory that is no longer accessible to the program and reclaims it and so allows it to be reused

Often, the user program has to stop running while the GC does its thing as the GC may move values around in memory to tidy up the free spaces

Memory Management

GC

Note, if we do nothing, the program will likely eventually run out of memory

A garbage collector is code, usually part of the language runtime, that periodically searches through memory for memory that is no longer accessible to the program and reclaims it and so allows it to be reused

Often, the user program has to stop running while the GC does its thing as the GC may move values around in memory to tidy up the free spaces

Though well-designed GC code runs very quickly, or, in a few systems, in parallel with the application code

Memory Management

GC or Manual

Languages with integrated GC include Lisp, Haskell, Java, JavaScript, Perl, Python, Go

Memory Management

GC or Manual

Languages with integrated GC include Lisp, Haskell, Java, JavaScript, Perl, Python, Go

Languages without integrated GC include C, C++, Rust

Memory Management

GC or Manual

Languages with integrated GC include Lisp, Haskell, Java, JavaScript, Perl, Python, Go

Languages without integrated GC include C, C++, Rust

In languages that use *manual memory management* (not GC) if you drop all references to an object, that's the programmer's problem and this is generally regarded as a bug as that memory is now permanently lost to the program

Memory Management

GC or Manual

Languages with integrated GC include Lisp, Haskell, Java, JavaScript, Perl, Python, Go

Languages without integrated GC include C, C++, Rust

In languages that use *manual memory management* (not GC) if you drop all references to an object, that's the programmer's problem and this is generally regarded as a bug as that memory is now permanently lost to the program

It happens that the Java-like code above is also valid C++

Memory Management

GC or Manual

Languages with integrated GC include Lisp, Haskell, Java, JavaScript, Perl, Python, Go

Languages without integrated GC include C, C++, Rust

In languages that use *manual memory management* (not GC) if you drop all references to an object, that's the programmer's problem and this is generally regarded as a bug as that memory is now permanently lost to the program

It happens that the Java-like code above is also valid C++

Thus it is buggy C++ with a *memory leak*: the memory is never recovered and reused

Memory Management

GC or Manual

With MMM the programmer needs to explicitly allocate memory for values and then free (deallocate) that memory when they are done with it

```
int *array = malloc(4*sizeof(int));  // allocate
...
... do stuff with array
...
free(array);  // deallocate
```

Memory Management

GC or Manual

With MMM the programmer needs to explicitly allocate memory for values and then free (deallocate) that memory when they are done with it

```
int *array = malloc(4*sizeof(int));  // allocate
...
... do stuff with array
...
free(array);  // deallocate
```

Remember, in real code, the allocate and free can be thousands of lines of code apart, and written by different programmers; or there might be more than one place where allocation or deallocation could be done; and so on

Memory Management

GC or Manual

If the free is forgotten (after we have finished with `array`), that's memory that can never be used again (a leak)

Memory Management

GC or Manual

If the free is forgotten (after we have finished with `array`), that's memory that can never be used again (a leak)

If the `array` is used after the free, that's a bug as the memory it uses might now have been allocated to something else in the system

Memory Management

GC or Manual

If the free is forgotten (after we have finished with `array`), that's memory that can never be used again (a leak)

If the `array` is used after the free, that's a bug as the memory it uses might now have been allocated to something else in the system

Writing into the array is overwriting the bytes used by some other value in the system

Memory Management

Manual

A program with a memory leak will gradually use more and more memory until the OS says it's had enough

Memory Management

Manual

A program with a memory leak will gradually use more and more memory until the OS says it's had enough

And then the program probably crashes as the programmer only tested it on small examples

Memory Management

Manual

A program with a memory leak will gradually use more and more memory until the OS says it's had enough

And then the program probably crashes as the programmer only tested it on small examples

In Java the `bigclass` example is arguably not buggy, but it is definitely poor code as it wastes time creating useless objects; and wastes time collecting the garbage

Memory Management

Manual

A program with a memory leak will gradually use more and more memory until the OS says it's had enough

And then the program probably crashes as the programmer only tested it on small examples

In Java the `bigclass` example is arguably not buggy, but it is definitely poor code as it wastes time creating useless objects; and wastes time collecting the garbage

Exercise Find the time it takes to create an object in Java, or your favourite language

Memory Management

Manual

Code written using manual memory management must be very careful on its use of memory, e.g., use of `malloc` and `free`, or `new` and `delete`, to ensure they match up correctly

Memory Management

Manual

Code written using manual memory management must be very careful on its use of memory, e.g., use of `malloc` and `free`, or `new` and `delete`, to ensure they match up correctly

That is to say, the *application programmer* must be very careful

Memory Management

Manual

Code written using manual memory management must be very careful on its use of memory, e.g., use of `malloc` and `free`, or `new` and `delete`, to ensure they match up correctly

That is to say, the *application programmer* must be very careful

Code written in GC languages can let the GC take care of things

Memory Management

Manual

Code written using manual memory management must be very careful on its use of memory, e.g., use of `malloc` and `free`, or `new` and `delete`, to ensure they match up correctly

That is to say, the *application programmer* must be very careful

Code written in GC languages can let the GC take care of things

That is to say, the *application programmer* doesn't have to take care

Memory Management

Manual

Code written using manual memory management must be very careful on its use of memory, e.g., use of `malloc` and `free`, or `new` and `delete`, to ensure they match up correctly

That is to say, the *application programmer* must be very careful

Code written in GC languages can let the GC take care of things

That is to say, the *application programmer* doesn't have to take care

(The programmer who implemented the GC in the runtime definitely needed to care!)

GC vs. Manual

GC: no memory worries for the programmer, but generally less efficient (as the GC has to search through memory to find inaccessible memory) and encourages sloppy programming

GC vs. Manual

GC: no memory worries for the programmer, but generally less efficient (as the GC has to search through memory to find inaccessible memory) and encourages sloppy programming

The memory system and GC code has to be generic and work in all kinds of situations

GC vs. Manual

GC: no memory worries for the programmer, but generally less efficient (as the GC has to search through memory to find inaccessible memory) and encourages sloppy programming

The memory system and GC code has to be generic and work in all kinds of situations

MMM: allows precise memory management as the programmer will explicitly allocate and deallocate memory, but also facilitates buggy programming as it is easy for the programmer to get it wrong

GC vs. Manual

GC: no memory worries for the programmer, but generally less efficient (as the GC has to search through memory to find inaccessible memory) and encourages sloppy programming

The memory system and GC code has to be generic and work in all kinds of situations

MMM: allows precise memory management as the programmer will explicitly allocate and deallocate memory, but also facilitates buggy programming as it is easy for the programmer to get it wrong

The programmer can tune the use of memory for their application

GC

A garbage collector can be added to C and C++ (etc.) as a library

GC

A garbage collector can be added to C and C++ (etc.) as a library

Not as precise as an in-built GC as (for various reasons): it might miss occasional bits of garbage

GC

A garbage collector can be added to C and C++ (etc.) as a library

Not as precise as an in-built GC as (for various reasons): it might miss occasional bits of garbage

Exercise Is this the best of both worlds or the worst of both worlds?

Memory Management

Maybe the best approach is to write good and correct code in the first place?

Memory Management

Maybe the best approach is to write good and correct code in the first place?

But that's never going to happen

Memory Management

Maybe the best approach is to write good and correct code in the first place?

But that's never going to happen

> *"Around 70 percent of all the vulnerabilities in Microsoft products addressed through a security update each year are memory safety issues"*
> Matt Miller, Microsoft security engineer, Feb 2019

Memory Management

Memory safety bugs in C and C++ continue to be the most-difficult-to-address source of incorrectness. We invest a great deal of effort and resources into detecting, fixing, and mitigating this class of bugs, and these efforts are effective in preventing a large number of bugs from making it into Android releases. Yet in spite of these efforts, memory safety bugs continue to be a top contributor of stability issues, and consistently represent approximately 70% of Android's high severity security vulnerabilities.

Jeff Vander Stoep and Stephen Hines, Android Team, April 2021

Memory Management

Manual memory management bugs include:

- use after free
- double free
- invalid free (passing a non-`malloc`ed pointer to `free`)
- no check for allocation fail
- memory leak

Memory Management

Manual memory management bugs include:

- use after free
- double free
- invalid free (passing a non-`malloc`ed pointer to `free`)
- no check for allocation fail
- memory leak

None of these are applicable to a GC language

Memory Management

Other memory errors include:

- reading uninitialised memory
- accessing beyond the bounds of allocated memory, e.g., beyond the ends of a vector

affecting both GC and MMM

Memory Management

Other memory errors include:

- reading uninitialised memory
- accessing beyond the bounds of allocated memory, e.g., beyond the ends of a vector

affecting both GC and MMM

Some languages have memory access checking to avoid these kinds of errors, but checking will slow the running of your program down

Memory Management

For example, `x[n] = 42;`

- Checked: the running code first checks to make sure that `n` is not beyond the ends of the vector; then it does the assignment

Memory Management

For example, `x[n] = 42;`

- Checked: the running code first checks to make sure that `n` is not beyond the ends of the vector; then it does the assignment
- Safe, but slower

Memory Management

For example, `x[n] = 42;`

- Checked: the running code first checks to make sure that `n` is not beyond the ends of the vector; then it does the assignment
- Safe, but slower
- Unchecked: the code just does the assignment

Memory Management

For example, `x[n] = 42;`

- Checked: the running code first checks to make sure that `n` is not beyond the ends of the vector; then it does the assignment
- Safe, but slower
- Unchecked: the code just does the assignment
- Faster, but allows bugs

Memory Management

C does unchecked vector accesses

Memory Management

C does unchecked vector accesses

- because it is fast

Memory Management

C does unchecked vector accesses

- because it is fast
- *very occasionally* it is what the programmer wanted: e.g., `x[-1] = 23;` is valid C, and *very occasionally* useful to careful programmers

Memory Management

C does unchecked vector accesses

- because it is fast
- *very occasionally* it is what the programmer wanted: e.g., `x[-1] = 23;` is valid C, and *very occasionally* useful to careful programmers
- (And C vectors don't include information about their length, anyway)

Memory Management

Python checks vector accesses

Memory Management

Python checks vector accesses

- The Python design philosophy is safety before speed

Memory Management

Python checks vector accesses

- The Python design philosophy is safety before speed

Exercise What does `x[-1]` do in Python?

Memory Management

Python checks vector accesses

- The Python design philosophy is safety before speed

Exercise What does `x[-1]` do in Python?

Exercise And Java, Rust, C++, . . . ?

Memory Management

We have a trade-off of speed against correctness

Memory Management

We have a trade-off of speed against correctness

So people have developed tools to check code in languages that don't check for themselves

Memory Management

We have a trade-off of speed against correctness

So people have developed tools to check code in languages that don't check for themselves

Some tools work at compile time; some at run time

Memory Management

Exercise Read about `valgrind`, `Allinea`, `AddressSanitizer` and other runtime memory checking tools

Memory Management

Exercise Read about `valgrind`, `Allinea`, `AddressSanitizer` and other runtime memory checking tools

Exercise Do such tools imply there is a problem with the design of the language?

Memory Management

Exercise Read about `valgrind`, `Allinea`, `AddressSanitizer` and other runtime memory checking tools

Exercise Do such tools imply there is a problem with the design of the language?

Exercise Some clever compilers use proofs to avoid the need for run-time checking. E.g., somehow it can prove that `n` cannot be too large in the access `x[n]`. Read about this

Memory Management

Exercise Read about `valgrind`, `Allinea`, `AddressSanitizer` and other runtime memory checking tools

Exercise Do such tools imply there is a problem with the design of the language?

Exercise Some clever compilers use proofs to avoid the need for run-time checking. E.g., somehow it can prove that `n` cannot be too large in the access `x[n]`. Read about this

Exercise Find out whether your favourite language checks or not; if it checks, what are the overheads?