So why do all these varieties of parallel architecture exist?
So why do all these varieties of parallel architecture exist?

There is essentially just one way uniprocessor machines are built: the von Neumann model
So why do all these varieties of parallel architecture exist?

There is essentially just one way uniprocessor machines are built: the von Neumann model

Is there a model that encapsulates multiprocessors in the same way?
So why do all these varieties of parallel architecture exist?

There is essentially just one way uniprocessor machines are built: the von Neumann model

Is there a model that encapsulates multiprocessors in the same way?

There are many contenders, but no obvious winner
We have the original von Neumann 5 box model
Classifications
Extensions of von Neumann

Shared memory MIMD

- ALU
- Control
- ALU
- Control
- ALU
- Control

memory
Classifications
Extensions of von Neumann

Distributed memory MIMD

Input

Output

ALU
Control

Memory

ALU
Control

Memory

ALU
Control

Memory
Classifications
Extensions of von Neumann

SIMD

input

ALU
memory

ALU
memory

ALU
memory

ALU
memory

ALU
memory

ALU
memory

output

Control

Memory
Perhaps there just isn’t a single extension of von Neumann that is suitable as a one-size-fits-all solution
Perhaps there just isn’t a single extension of von Neumann that is suitable as a one-size-fits-all solution

Or perhaps we just haven’t thought of it yet?
There are several theoretical models whose aim is to guide the design of parallel algorithms and allow the analysis of them.
There are several theoretical models whose aim is to guide the design of parallel algorithms and allow the analysis of them.

As with von Neumann, the idea is that you

- write your program in accordance with the model
- the model maps well onto all kinds of real hardware
- therefore your program maps well onto all kinds of real hardware
Classifications
Extensions of von Neumann

PRAM

The *Parallel Random Access Machine* model idealises a parallel computer as shared memory MIMD, concentrating on the memory bottleneck.
Classifications
Extensions of von Neumann

PRAM

The *Parallel Random Access Machine* model idealises a parallel computer as shared memory MIMD, concentrating on the memory bottleneck

You have a choice of how memory can be accessed:
Classifications
Extensions of von Neumann

PRAM

The *Parallel Random Access Machine* model idealises a parallel computer as shared memory MIMD, concentrating on the memory bottleneck.

You have a choice of how memory can be accessed:

- *Exclusive Read Exclusive Write* (EREW). Each memory location can only be read or written by *one* processor at a time. The simplest architecture
Classifications
Extensions of von Neumann

PRAM

The *Parallel Random Access Machine* model idealises a parallel computer as shared memory MIMD, concentrating on the memory bottleneck.

You have a choice of how memory can be accessed:

- **Exclusive Read Exclusive Write** (EREW). Each memory location can only be read or written by *one* processor at a time. The simplest architecture.

- **Concurrent Read Exclusive Write** (CREW). Each memory location can be read by many processors simultaneously, but written by just *one* processor at a time (c.f. global memory in a vector processor).
Classifications
Extensions of von Neumann

PRAM

- *Concurrent Read Concurrent Write* (CRCW). Each memory location can be read or written by *many* processors simultaneously. Not a realistic model.
Classifications
Extensions of von Neumann

PRAM

- *Concurrent Read Concurrent Write* (CRCW). Each memory location can be read or written by *many* processors simultaneously. Not a realistic model.
- *Exclusive Read Concurrent Write* (ERCW). The fourth combination, never used.
PRAM

PRAMs make many further simplifying assumptions, including:

- Memory is symmetric: every location is accessed at the same speed: decreasingly realistic
- There are an unlimited number of processors: there's always another processor if you need it. Seems unrealistic, but not so bad as you think as most programs are unable to make use of the hardware that we currently have
- Memory is unlimited. This assumption is also often made in analysis of uniprocessor algorithms
PRAM

PRAMs make many further simplifying assumptions, including:

- Memory is symmetric: every location is accessed at the same speed: decreasingly realistic
PRAM

PRAMs make many further simplifying assumptions, including:

- Memory is symmetric: every location is accessed at the same speed: decreasingly realistic
- There are an unlimited number of processors: there’s always another processor if you need it. Seems unrealistic, but not so bad as you think as most programs are unable to make use of the hardware that we currently have
PRAM

PRAMs make many further simplifying assumptions, including:

- Memory is symmetric: every location is accessed at the same speed: decreasingly realistic
- There are an unlimited number of processors: there’s always another processor if you need it. Seems unrealistic, but not so bad as you think as most programs are unable to make use of the hardware that we currently have
- Memory is unlimited. This assumption is also often made in analysis of uniprocessor algorithms
Classifications
Extensions of von Neumann

In the early days of Computer Science, there were many clever algorithms invented to deal with the lack of available memory.
In the early days of Computer Science, there were many clever algorithms invented to deal with the lack of available memory.

And, to some extent, memory is still limited in some modern architectures that have very large numbers of CPUs but each with only a small amount of memory.
Classifications
Extensions of von Neumann

PRAM

So you analyse your program, counting numbers of memory accesses (according to EREW/CREW/CRCW) and this gives you a measure of the time your program will take to run
Classifications
Extensions of von Neumann

PRAM

So you analyse your program, counting numbers of memory accesses (according to EREW/CREW/CRCW) and this gives you a measure of the time your program will take to run.

This is primarily a MIMD model, but you can analyse SIMD using it.
Classifications
Extensions of von Neumann

PRAM

So you analyse your program, counting numbers of memory accesses (according to EREW/CREW/CRCW) and this gives you a measure of the time your program will take to run.

This is primarily a MIMD model, but you can analyse SIMD using it.

It totally ignores important realities like NUMA and other overheads, such as communication time in a distributed memory system.
PRAM

So you analyse your program, counting numbers of memory accesses (according to EREW/CREW/CRCW) and this gives you a measure of the time your program will take to run

This is primarily a MIMD model, but you can analyse SIMD using it

It totally ignores important realities like NUMA and other overheads, such as communication time in a distributed memory system

But it gives you a rough idea and it is extensively used in analysis of parallel algorithms: we’ll have some examples later
Classifications
Extensions of von Neumann

BSP

Another model, the *Bulk Synchronous Parallel* model
Classifications
Extensions of von Neumann

BSP

Another model, the *Bulk Synchronous Parallel* model

This model takes communication time into account
Classifications
Extensions of von Neumann

BSP

Another model, the *Bulk Synchronous Parallel* model

This model takes communication time into account

It assumes processors with local memory communicating over a network
BSP

Another model, the *Bulk Synchronous Parallel* model

This model takes communication time into account

It assumes processors with local memory communicating over a network

Good for distributed, but also for shared memory where you just have smaller communication costs
Classifications

Extensions of von Neumann

BSP

A computation is modelled as a sequence of *supersteps*
Classifications
Extensions of von Neumann

BSP

A computation is modelled as a sequence of *supersteps*

- each processor does some computation (MIMD, but could be SIMD)
- each processor does some communication
- each processors waits at a global *barrier* until everybody has finished their communications. This is the “bulk synchronous” part
- repeat
Classifications
Extensions of von Neumann

BSP
Classifications
Extensions of von Neumann

BSP

Processing is simplified in this way to give us a chance of being able to make an analysis.
Classifications
Extensions of von Neumann

BSP

Processing is simplified in this way to give us a chance of being able to make an analysis

Fortunately, many real computations are not too far from this shape
Classifications
Extensions of von Neumann

BSP

Processing is simplified in this way to give us a chance of being able to make an analysis

Fortunately, many real computations are not too far from this shape

More realistic than PRAMs, but harder work to get analyses out of it
Classifications
Extensions of von Neumann

BSP

Processing is simplified in this way to give us a chance of being able to make an analysis

Fortunately, many real computations are not too far from this shape

More realistic than PRAMs, but harder work to get analyses out of it

But those analyses tend to be a better match to realistic hardware
And so on for many other models, some practical, some not (e.g., parallel Turing machines)
Classifications
Extensions of von Neumann

And so on for many other models, some practical, some not (e.g., parallel Turing machines)

But the fact remains that there is not one simple model that works for all kinds of parallel processing
And so on for many other models, some practical, some not (e.g., parallel Turing machines)

But the fact remains that there is not one simple model that works for all kinds of parallel processing

This is probably the source of the confusion in parallel hardware, but we have to live with it