Classifications

Shared Memory

Symmetric shared memory is the model that current small machines (multicore PCs) use

Classifications

Shared Memory

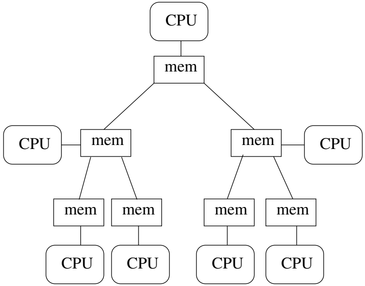
Symmetric shared memory is the model that current small machines (multicore PCs) use

It is well suited to MIMD, but note that SIMD also uses symmetric shared memory, but with a different access pattern

Classifications

NUMA

So if symmetric, i.e., uniform access, shared memory does not scale, we can try managing memory in other ways



Example NUMA

Each processor has a chunk of memory, but can also access memory of other processors, perhaps arranged in a tree

Classifications

NUMA

A processor will have fast access to its closest chunk of memory, but slower access to more remote memory

Classifications

NUMA

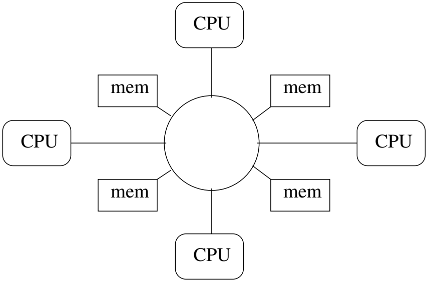
A processor will have fast access to its closest chunk of memory, but slower access to more remote memory

And different chunks of remote memory will have different access speeds

Classifications

NUMA

Of course many other topologies have been tried: star, ring, hypercube, full interconnect, and so on

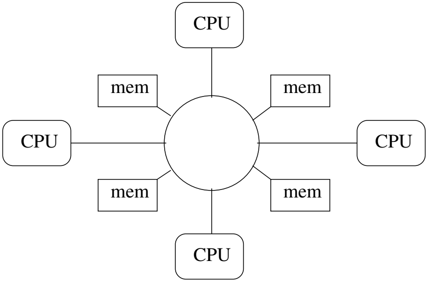


Memory in a ring

Classifications

NUMA

Of course many other topologies have been tried: star, ring, hypercube, full interconnect, and so on



Memory in a ring

This architecture evens out the access time to different chunks of memory a little

Classifications

NUMA

These are *non-uniform memory access*

Classifications

NUMA

These are *non-uniform memory access*

NUMA shared memory scales much better than symmetric shared memory

Classifications

NUMA

These are *non-uniform memory access*

NUMA shared memory scales much better than symmetric shared memory

By scaling here we mean you can build larger machines with more processors cost effectively

Classifications

NUMA

These are *non-uniform memory access*

NUMA shared memory scales much better than symmetric shared memory

By scaling here we mean you can build larger machines with more processors cost effectively

But there is a downside: now programs and programmers (and the OS) have to worry about *data locality*: data a processor needs should be kept close to that processor

Classifications

NUMA

These are *non-uniform memory access*

NUMA shared memory scales much better than symmetric shared memory

By scaling here we mean you can build larger machines with more processors cost effectively

But there is a downside: now programs and programmers (and the OS) have to worry about *data locality*: data a processor needs should be kept close to that processor

It can make a huge difference to the speed of a program if the data is not where it should be

Classifications

NUMA

If data is close to the processor that is using it, it will go faster than if the data has to be fetched from further away

Classifications

NUMA

If data is close to the processor that is using it, it will go faster than if the data has to be fetched from further away

So you try to keep data near the relevant processor

Classifications

NUMA

If data is close to the processor that is using it, it will go faster than if the data has to be fetched from further away

So you try to keep data near the relevant processor

Or the computation on a processor near to the data

Classifications

NUMA

If data is close to the processor that is using it, it will go faster than if the data has to be fetched from further away

So you try to keep data near the relevant processor

Or the computation on a processor near to the data

Of course, if data needs to be used by several processors, this becomes a very difficult scheduling problem

Classifications

NUMA

NUMA implementations stratify the memory in terms of “distance”

Classifications

NUMA

NUMA implementations stratify the memory in terms of "distance"

For example:

- direct connection on the local memory bus
- on the same node
- one hop away
- two hops away
- and so on

Classifications

NUMA

Though this is often simplified to: local, remote, and “far away”

Classifications

NUMA

Though this is often simplified to: local, remote, and “far away”

The OS or system libraries or the programmer will try their best to place data in appropriate memory to minimise latency, using their knowledge of the NUMA hierarchy and their knowledge of the program’s needs

Classifications

NUMA

Though this is often simplified to: local, remote, and “far away”

The OS or system libraries or the programmer will try their best to place data in appropriate memory to minimise latency, using their knowledge of the NUMA hierarchy and their knowledge of the program’s needs

The programmer ideally would have a good idea of the architecture of a machine before writing code for it

Classifications

NUMA

Though this is often simplified to: local, remote, and “far away”

The OS or system libraries or the programmer will try their best to place data in appropriate memory to minimise latency, using their knowledge of the NUMA hierarchy and their knowledge of the program’s needs

The programmer ideally would have a good idea of the architecture of a machine before writing code for it

And so the portability of a program is in question

Classifications

NUMA

Though this is often simplified to: local, remote, and “far away”

The OS or system libraries or the programmer will try their best to place data in appropriate memory to minimise latency, using their knowledge of the NUMA hierarchy and their knowledge of the program’s needs

The programmer ideally would have a good idea of the architecture of a machine before writing code for it

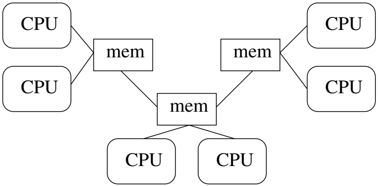
And so the portability of a program is in question

This is still a matter of great research and development!

Classifications

NUMA

And, of course, there are hybrids where CPUs share some memory symmetrically and some memory NUMA



Hybrid NUMA

Classifications

Distributed Memory

NUMA allows architectures to scale to greater numbers of processors, but it won’t scale indefinitely, perhaps a few 1000s of cores

Classifications

Distributed Memory

NUMA allows architectures to scale to greater numbers of processors, but it won't scale indefinitely, perhaps a few 1000s of cores

If the problem is the memory bus bottleneck which means you have to keep cached copies of a value, and then you have the problem of keeping coherence amongst the copies, why not simply *not* have shared memory?

Classifications

Distributed Memory

NUMA allows architectures to scale to greater numbers of processors, but it won't scale indefinitely, perhaps a few 1000s of cores

If the problem is the memory bus bottleneck which means you have to keep cached copies of a value, and then you have the problem of keeping coherence amongst the copies, why not simply *not* have shared memory?

Distributed memory says each processor's memory it its own and is entirely separate from every other processor's memory

Classifications

Distributed Memory

Shared memory processors share a *single memory address space*: within a single process memory location 42 on one processor refers to the same thing as memory location 42 on every other processor, as it's the same memory

Classifications

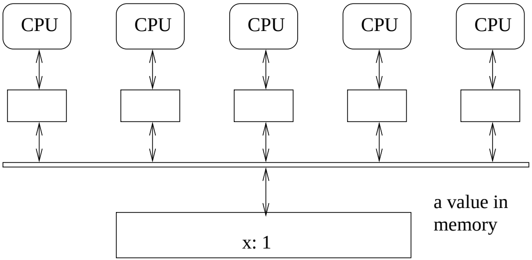
Distributed Memory

Shared memory processors share a *single memory address space*: within a single process memory location 42 on one processor refers to the same thing as memory location 42 on every other processor, as it's the same memory

The variable `x` on this processor is the same as the `x` on that processor (assuming SPMD)

Classifications

Distributed Memory



Shared address space

Classifications

Distributed Memory

Processors in a distributed memory architecture each have their own, separate, address space

Classifications

Distributed Memory

Processors in a distributed memory architecture each have their own, separate, address space

Memory location 42 on one processor is entirely separate from memory location 42 on every other processor

Classifications

Distributed Memory

Processors in a distributed memory architecture each have their own, separate, address space

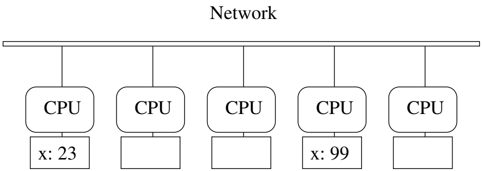
Memory location 42 on one processor is entirely separate from memory location 42 on every other processor

Each processor has their own version of variable `x`, nothing to do with any other `x` on other processors

Classifications

Distributed Memory

Each processor has its own memory

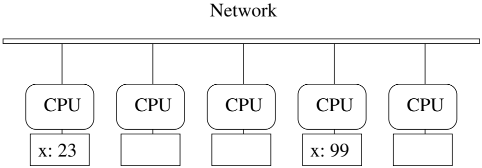


Distributed memory architecture

Classifications

Distributed Memory

Each processor has its own memory



Distributed memory architecture

Typically connected by a network, rather than an expensive memory bus

Classifications

Distributed Memory

To get at data on another node a processor sends a *message* to that node, which will reply with the data

Classifications

Distributed Memory

To get at data on another node a processor sends a *message* to that node, which will reply with the data

Clearly this *message passing* will be very much slower than simple shared memory accesses

Classifications

Distributed Memory

To get at data on another node a processor sends a *message* to that node, which will reply with the data

Clearly this *message passing* will be very much slower than simple shared memory accesses

Memory access across a network can be factors of thousands, perhaps millions times slower than local memory

Classifications

Distributed Memory

To get at data on another node a processor sends a *message* to that node, which will reply with the data

Clearly this *message passing* will be very much slower than simple shared memory accesses

Memory access across a network can be factors of thousands, perhaps millions times slower than local memory

The position of data is now *very* important

Classifications

Distributed Memory

To get at data on another node a processor sends a *message* to that node, which will reply with the data

Clearly this *message passing* will be very much slower than simple shared memory accesses

Memory access across a network can be factors of thousands, perhaps millions times slower than local memory

The position of data is now *very* important

Your code has to change, too

Classifications

Distributed Memory

Think of a shared memory operation:

`x = y;`

`x` gets the value of `y`, “simply” read from memory

Classifications

Distributed Memory

Think of a shared memory operation:

`x = y;`

`x` gets the value of `y`, "simply" read from memory

Compared with the overhead in distributed memory of creating a message, sending, waiting, reading the reply

Classifications

Distributed Memory

Think of a shared memory operation:

`x = y;`

`x` gets the value of `y`, “simply” read from memory

Compared with the overhead in distributed memory of creating a message, sending, waiting, reading the reply

See MPI (Message Passing Interface) later, but conceptually we have to write

```
x = FetchDouble(remotecpuname, "y");
```

Classifications

Distributed Memory

Some underlying message passing library does the hard work of the messaging

Classifications

Distributed Memory

Some underlying message passing library does the hard work of the messaging

Your code become much more complex to write

Classifications

Distributed Memory

Some underlying message passing library does the hard work of the messaging

Your code become much more complex to write

Both in needing a lot more text, and in needing thought on where to put your data

Classifications

Message Passing

Note that you can also use message passing on a shared memory architecture

Classifications

Message Passing

Note that you can also use message passing on a shared memory architecture

Doing so might be useful for coding or program structure reasons

Classifications

Message Passing

Note that you can also use message passing on a shared memory architecture

Doing so might be useful for coding or program structure reasons

The underlying messages are now probably implemented as simple accesses to shared memory

Classifications

Message Passing

Note that you can also use message passing on a shared memory architecture

Doing so might be useful for coding or program structure reasons

The underlying messages are now probably implemented as simple accesses to shared memory

Some parallel programming systems (see later) *only* provide messaging across threads (often via mechanisms called *channels*), thus masking the underlying architecture and improving program portability across architectures

Classifications

Distributed Memory

When using distributed memory you try to keep the data a process needs on the processor it is running on, maybe even replicating data or replicating computations, and access remote data as little as you get away with

Classifications

Distributed Memory

When using distributed memory you try to keep the data a process needs on the processor it is running on, maybe even replicating data or replicating computations, and access remote data as little as you get away with

You have to balance the cost of the computations against the cost of the data movement

Classifications

Distributed Memory

When using distributed memory you try to keep the data a process needs on the processor it is running on, maybe even replicating data or replicating computations, and access remote data as little as you get away with

You have to balance the cost of the computations against the cost of the data movement

An ideal that is rarely achieved in real programs

Classifications

Distributed Memory

When using distributed memory you try to keep the data a process needs on the processor it is running on, maybe even replicating data or replicating computations, and access remote data as little as you get away with

You have to balance the cost of the computations against the cost of the data movement

An ideal that is rarely achieved in real programs

Of course, if you replicate data that gets updated, you immediately have a coherence problem again, but now your own code has to deal with it

Classifications

Distributed Memory

Note that replicating read-only data (e.g., tables of values) will be fine: there is no coherence issue with multiple copies of data that never changes

Classifications

Distributed Memory

Note that replicating read-only data (e.g., tables of values) will be fine: there is no coherence issue with multiple copies of data that never changes

But you do need to put a lot of thought into replicating read-write (mutable) data

Classifications

DMA

More sophisticated systems have extensive hardware support for messaging

Classifications

DMA

More sophisticated systems have extensive hardware support for messaging

They have specific *direct memory access* (DMA) hardware that accesses memory independently of the CPUs

Classifications

DMA

More sophisticated systems have extensive hardware support for messaging

They have specific *direct memory access* (DMA) hardware that accesses memory independently of the CPUs

Thus messaging proceeds independently of the CPU: communication is *asynchronous* with computation, freeing the CPU to do something else while the message is being processed by the DMA hardware

Classifications

DMA

More sophisticated systems have extensive hardware support for messaging

They have specific *direct memory access* (DMA) hardware that accesses memory independently of the CPUs

Thus messaging proceeds independently of the CPU: communication is *asynchronous* with computation, freeing the CPU to do something else while the message is being processed by the DMA hardware

Thus allowing more computation; but at the cost of more complicated programming

Classifications

Computation vs. Communication

The call to `FetchDouble` above could return immediately and allow your code to continue computing on something else, rather than waiting for the value of `y` to appear: but you can't use `x` until the value has arrived some time later

Classifications

Computation vs. Communication

The call to `FetchDouble` above could return immediately and allow your code to continue computing on something else, rather than waiting for the value of `y` to appear: but you can't use `x` until the value has arrived some time later

Of course, you now need some mechanism to be notified when the value *has* arrived, and so you can now use `x`

Classifications

Computation vs. Communication

The call to `FetchDouble` above could return immediately and allow your code to continue computing on something else, rather than waiting for the value of `y` to appear: but you can't use `x` until the value has arrived some time later

Of course, you now need some mechanism to be notified when the value *has* arrived, and so you can now use `x`

Such asynchronous programming is very hard to get right

Classifications

Computation vs. Communication

The call to `FetchDouble` above could return immediately and allow your code to continue computing on something else, rather than waiting for the value of `y` to appear: but you can't use `x` until the value has arrived some time later

Of course, you now need some mechanism to be notified when the value *has* arrived, and so you can now use `x`

Such asynchronous programming is very hard to get right

But this idea of overlapping computation and communication is important and will reappear many times

Classifications

Distributed Memory

In distributed systems the concept of single shared values has to go completely out of the window

Classifications

Distributed Memory

In distributed systems the concept of single shared values has to go completely out of the window

The value of `x` here is nothing to do with the value of `x` there

Classifications

Distributed Memory

In distributed systems the concept of single shared values has to go completely out of the window

The value of `x` here is nothing to do with the value of `x` there

Programs have to be written with this in mind: global shared mutable values are simply not a good idea, even in uniprocessor programs!

Classifications

Distributed Memory

Distributed memory is the architecture used by clusters: each node is effectively a PC

Classifications

Distributed Memory

Distributed memory is the architecture used by clusters: each node is effectively a PC

Very suitable for SPMD, not so suitable for SIMD

Classifications

Distributed Memory

Distributed memory is the architecture used by clusters: each node is effectively a PC

Very suitable for SPMD, not so suitable for SIMD

Even with the huge message passing overhead, clusters are very popular, particularly with very large problems where the overhead is small relative to the rest of the computation

Classifications

Distributed Memory

Distributed memory is the architecture used by clusters: each node is effectively a PC

Very suitable for SPMD, not so suitable for SIMD

Even with the huge message passing overhead, clusters are very popular, particularly with very large problems where the overhead is small relative to the rest of the computation

The computations do have to be huge!

Classifications

Distributed Memory

Not suitable for small problems, or problems where data need to move a lot between processors

Classifications

Distributed Memory

Not suitable for small problems, or problems where data need to move a lot between processors

Scales *very* well as an architecture. Clusters of over a million cores exist: see the TOP500 list

Classifications

Scaling

Making big machines is easier with distributed systems, too

Classifications

Scaling

Making big machines is easier with distributed systems, too

When we try to add CPUs to a shared memory system, we have to pay a great deal for the complicated memory architecture as it means redesigning the silicon and building new chips

Classifications

Scaling

Making big machines is easier with distributed systems, too

When we try to add CPUs to a shared memory system, we have to pay a great deal for the complicated memory architecture as it means redesigning the silicon and building new chips

This can quickly swamp all other costs, so making scaling a shared memory system impractical

Classifications

Scaling

Making big machines is easier with distributed systems, too

When we try to add CPUs to a shared memory system, we have to pay a great deal for the complicated memory architecture as it means redesigning the silicon and building new chips

This can quickly swamp all other costs, so making scaling a shared memory system impractical

In contrast, the cost of adding CPUs to a distributed memory system is “simply” the cost of the CPUs and the networking

Classifications

Scaling

Making big machines is easier with distributed systems, too

When we try to add CPUs to a shared memory system, we have to pay a great deal for the complicated memory architecture as it means redesigning the silicon and building new chips

This can quickly swamp all other costs, so making scaling a shared memory system impractical

In contrast, the cost of adding CPUs to a distributed memory system is “simply” the cost of the CPUs and the networking

This is roughly linear (per CPU) price scaling

Classifications

Distributed Memory

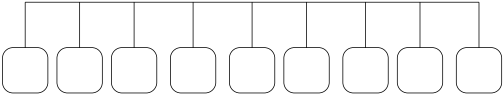
However, when scaling a cluster we should take care to scale the network, too, otherwise we have exactly the same kinds of bottleneck issues that shared memory systems have

Classifications

Distributed Memory

However, when scaling a cluster we should take care to scale the network, too, otherwise we have exactly the same kinds of bottleneck issues that shared memory systems have

In a network like



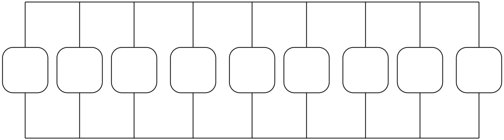
Simple shared network

the single shared network is clearly a bottleneck

Classifications

Distributed Memory

So we need to scale the network. There are many choices:



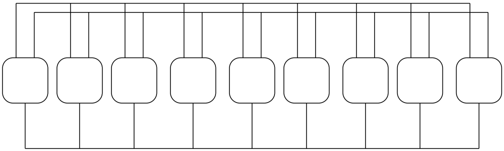
Network with two interfaces

Each processor would use one interface to communicate with processors 0, 2, 4, etc.., and the other interface to processors 1, 3, 5, etc., thus spreading the load

Classifications

Distributed Memory

Or three interfaces

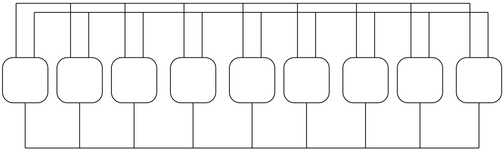


Network with three interfaces

Classifications

Distributed Memory

Or three interfaces



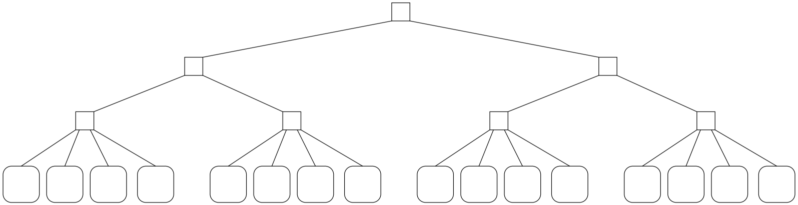
Network with three interfaces

But this gets expensive very quickly

Classifications

Distributed Memory

Trees are a good way of connecting things:

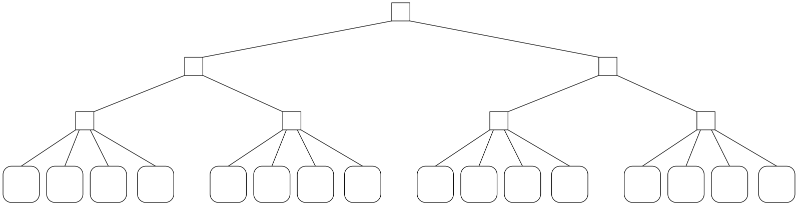


Tree network

Classifications

Distributed Memory

Trees are a good way of connecting things:

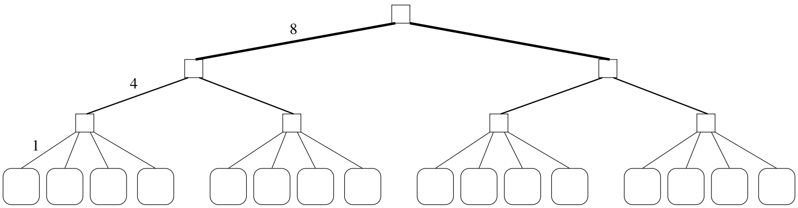


Tree network

Though the upper links now are a bottleneck, and we have introduced another non-uniformity

Classifications

Distributed Memory



Fat Tree

In a *fat tree* links up the tree have larger bandwidths, thus allowing full simultaneous bandwidth between each pair of nodes

Classifications

Distributed Memory

Though the latency between nodes will vary

Classifications

Distributed Memory

Though the latency between nodes will vary

In practice, a full fat tree is quite expensive, so real fat trees tend to skimp on the upper links a bit, e.g, 1, 2, 2 in the above diagram would be much cheaper to build (and a “2” would probably be a pair of “1”s)

Classifications

Distributed Memory

Though the latency between nodes will vary

In practice, a full fat tree is quite expensive, so real fat trees tend to skimp on the upper links a bit, e.g, 1, 2, 2 in the above diagram would be much cheaper to build (and a “2” would probably be a pair of “1”s)

Thus trading bandwidth for cost

Classifications

Distributed Memory

Though the latency between nodes will vary

In practice, a full fat tree is quite expensive, so real fat trees tend to skimp on the upper links a bit, e.g, 1, 2, 2 in the above diagram would be much cheaper to build (and a “2” would probably be a pair of “1”s)

Thus trading bandwidth for cost

Many other topologies exist, such as hypercube, torus, Banyan, etc.

Classifications

Distributed Memory

Though the latency between nodes will vary

In practice, a full fat tree is quite expensive, so real fat trees tend to skimp on the upper links a bit, e.g, 1, 2, 2 in the above diagram would be much cheaper to build (and a “2” would probably be a pair of “1”s)

Thus trading bandwidth for cost

Many other topologies exist, such as hypercube, torus, Banyan, etc.

Exercise Azure uses a *Clos network* within its datacentres. Read about this

Classifications

Distributed Memory

The point here is that this is relatively cheap to do with a distributed memory network. But adding bandwidth by doing this kind of connectivity in a shared memory system is extremely expensive as it likely needs new silicon

Classifications

Distributed Memory

The point here is that this is relatively cheap to do with a distributed memory network. But adding bandwidth by doing this kind of connectivity in a shared memory system is extremely expensive as it likely needs new silicon

Adding bandwidth in a network is relatively cheap

Classifications

Distributed Memory

The point here is that this is relatively cheap to do with a distributed memory network. But adding bandwidth by doing this kind of connectivity in a shared memory system is extremely expensive as it likely needs new silicon

Adding bandwidth in a network is relatively cheap

But decreasing latency is very expensive whatever the system

Classifications

Virtual Shared/Distributed Virtual Memory

Some programmers don't like the fact that distributed memory machines require programming using message passing and prefer the shared address space model: shared memory is easier to write programs for (they claim)

Classifications

Virtual Shared/Distributed Virtual Memory

Some programmers don’t like the fact that distributed memory machines require programming using message passing and prefer the shared address space model: shared memory is easier to write programs for (they claim)

They can use *virtual shared memory*

Classifications

Virtual Shared/Distributed Virtual Memory

Some programmers don't like the fact that distributed memory machines require programming using message passing and prefer the shared address space model: shared memory is easier to write programs for (they claim)

They can use *virtual shared memory*

Just as virtual memory is a way of converting virtual memory addresses into physical memory addresses, virtual shared memory is a mechanism to have a single, virtual, address space that is converted into distributed physical addresses

Classifications

Virtual Shared/Distributed Virtual Memory

Some programmers don't like the fact that distributed memory machines require programming using message passing and prefer the shared address space model: shared memory is easier to write programs for (they claim)

They can use *virtual shared memory*

Just as virtual memory is a way of converting virtual memory addresses into physical memory addresses, virtual shared memory is a mechanism to have a single, virtual, address space that is converted into distributed physical addresses

Thus this is also called *distributed virtual memory* and *distributed shared memory*

Classifications

Virtual Shared/Distributed Virtual Memory

Reading and writing variables will be implemented by a message passing layer hidden from the programmer in the OS or systems libraries

Classifications

Virtual Shared/Distributed Virtual Memory

Reading and writing variables will be implemented by a message passing layer hidden from the programmer in the OS or systems libraries

So the programmer won't have to care about it and they can write programs as if the whole of memory was one big chunk

Classifications

Virtual Shared/Distributed Virtual Memory

Reading and writing variables will be implemented by a message passing layer hidden from the programmer in the OS or systems libraries

So the programmer won't have to care about it and they can write programs as if the whole of memory was one big chunk

The programmer writes the simple "`x = y`" and the compiler/OS converts this into a shared memory access or a message call as appropriate

Classifications

Virtual Shared/Distributed Virtual Memory

Reading and writing variables will be implemented by a message passing layer hidden from the programmer in the OS or systems libraries

So the programmer won't have to care about it and they can write programs as if the whole of memory was one big chunk

The programmer writes the simple "`x = y`" and the compiler/OS converts this into a shared memory access or a message call as appropriate

But it will be very NUMA to data

Classifications

Virtual Shared/Distributed Virtual Memory

Unfortunately, programmers *do* have to care as the speed of a program will be very hard to predict or control, depending on how data is distributed across memory and the particular NUMA architecture it is running on

Classifications

Virtual Shared/Distributed Virtual Memory

Unfortunately, programmers *do* have to care as the speed of a program will be very hard to predict or control, depending on how data is distributed across memory and the particular NUMA architecture it is running on

How long does the assignment "`x = y`" take? Is it different from "`x = z`"?

Classifications

Virtual Shared/Distributed Virtual Memory

Unfortunately, programmers *do* have to care as the speed of a program will be very hard to predict or control, depending on how data is distributed across memory and the particular NUMA architecture it is running on

How long does the assignment "`x = y`" take? Is it different from "`x = z`"?

A good programmer looking for a good, consistent performance from their code will still need to think hard

Classifications

Virtual Shared/Distributed Virtual Memory

Unfortunately, programmers *do* have to care as the speed of a program will be very hard to predict or control, depending on how data is distributed across memory and the particular NUMA architecture it is running on

How long does the assignment "`x = y`" take? Is it different from "`x = z`"?

A good programmer looking for a good, consistent performance from their code will still need to think hard

A poor programmer will think their life is easier

Classifications

Virtual Shared/Distributed Virtual Memory

The underlying system also needs to solve all the problems of cache coherence that shared memory hardware has, but now using the (relatively) slow messaging passing layer rather than custom-designed hardware

Classifications

Virtual Shared/Distributed Virtual Memory

The underlying system also needs to solve all the problems of cache coherence that shared memory hardware has, but now using the (relatively) slow messaging passing layer rather than custom-designed hardware

The NUMA aspect is so unpredictable that many programmers prefer to be in control and have an explicitly non-shared model

Classifications

Virtual Shared/Distributed Virtual Memory

The underlying system also needs to solve all the problems of cache coherence that shared memory hardware has, but now using the (relatively) slow messaging passing layer rather than custom-designed hardware

The NUMA aspect is so unpredictable that many programmers prefer to be in control and have an explicitly non-shared model

When you write `FetchDouble` you *know* it is going to be slow

Classifications

Virtual Shared/Distributed Virtual Memory

The underlying system also needs to solve all the problems of cache coherence that shared memory hardware has, but now using the (relatively) slow messaging passing layer rather than custom-designed hardware

The NUMA aspect is so unpredictable that many programmers prefer to be in control and have an explicitly non-shared model

When you write `FetchDouble` you *know* it is going to be slow

Compare with "how fast is `x = y`?" in VSM

Classifications

Virtual Shared/Distributed Virtual Memory

The underlying communications layer in VSM might be implemented

Classifications

Virtual Shared/Distributed Virtual Memory

The underlying communications layer in VSM might be implemented

- in the Operating System, such as *Mosix*. This means all standard system libraries and user code can be used unchanged and a cluster looks like a single big machine: a *single system image* (SSI)

Classifications

Virtual Shared/Distributed Virtual Memory

The underlying communications layer in VSM might be implemented

- in the Operating System, such as *Mosix*. This means all standard system libraries and user code can be used unchanged and a cluster looks like a single big machine: a *single system image* (SSI)
- by the programming language and libraries, such as Cluster OpenMP or Unified Parallel C (see later), so the language may need a bit of learning by the programmer

Classifications

Virtual Shared/Distributed Virtual Memory

VSM is currently fairly rare in practice, though as NUMA techniques improve, people are starting to talk about *shared memory clusters* as being a viable and useful way to proceed

Latency numbers every programmer should know

L1 Cache hit	0.5 ns	0.5 sec one heart beat
Mutex lock/unlock	25 ns	25 sec making coffee
Main memory access	100 ns	100 sec brushing your teeth
Read 1MB from memory	250,000 ns	2.9 days a long weekend
Round trip within datacentre	500,000 ns	5.8 days a short holiday
Read 1MB from disk	30,000,000 ns	1 year
Send a packet California → Netherlands → California	150,000,000 ns	4.8 years two round trips to Mars

`https://gist.github.com/hellerbarde/2843375`

Classifications

The next class of architecture is one we have already touched on

Classifications

The next class of architecture is one we have already touched on

It has elements of both shared and distributed memory

Classifications

The next class of architecture is one we have already touched on

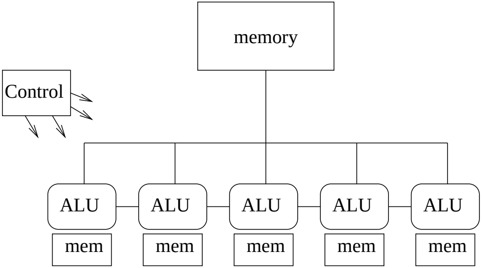
It has elements of both shared and distributed memory

It is used for data parallel computation

Classifications

Vectors

A *vector processor* is a SIMD collection of CPUs (actually ALUs), often with a chunk of global shared memory (and a single control unit)



Vector processor

Each processor also has its own chunk of local memory that it operates on

Classifications

Vectors

The local memory allows each ALU to work on a different set of values

Classifications

Vectors

The local memory allows each ALU to work on a different set of values

Note: this is *not* cache, but simply per-ALU memory

Classifications

Cache vs Local

Cache memory: a fast local copy of a slower memory location. If a value of a variable is cached on different cores, we want all the caches to contain the same value for that variable

Classifications

Cache vs Local

Cache memory: a fast local copy of a slower memory location. If a value of a variable is cached on different cores, we want all the caches to contain the same value for that variable

Local memory: per core memory (not always fast, by the way!) where we expect to have different values for a given variable in each