Hardware Architectures

Parallel Programming

2024-10-28

Prof. Dr. Michael Kuhn

michael.kuhn@ovgu.de

Parallel Computing and I/O
Institute for Intelligent Cooperating Systems
Faculty of Computer Science
Otto von Guericke University Magdeburg
https://parcio.ovgu.de

Outline

Hardware Architectures

- Review
- Introduction
- Classification
- Memory Access Models
- Network
- Summary

Performance Analysis and Optimization

- Why should performance measurements be repeated?
    1. Warm up the hardware
    2. Eliminate random errors
    3. Eliminate systematic errors
    4. Increase measurement time

- What does memory-bound stand for?
  1. Performance is limited by processor throughput
  2. Application does not have enough main memory
  3. Data cannot be moved in main memory
  4. Performance is limited by memory throughput

- Why are caches important?
  1. Decrease latency of memory accesses
  2. Increase size of main memory
  3. Reduce power consumption of memory accesses
  4. Increase throughput of main memory

- Which is the best performance metric?
  1. Runtime
  2. Utilization
  3. FLOPS
  4. FLOPS per watt

Outline

Hardware Architectures

- Review
- Introduction
- Classification
- Memory Access Models
- Network
- Summary

Motivation

- Parallel computing uses multiple similar components to solve problems faster
  - It can also be used to solve larger problems
  - For example, higher resolutions in climate models
- Multiple processors can process more instructions in the same time
  - More main memory or storage can be used to process larger problems
- Not all problems can be reasonably parallelized
  - One worker can dig a 1 m × 1 m × 1 m large hole in one hour

Motivation

- Parallel computing uses multiple similar components to solve problems faster
    - It can also be used to solve larger problems
    - For example, higher resolutions in climate models
- Multiple processors can process more instructions in the same time
    - More main memory or storage can be used to process larger problems
- Not all problems can be reasonably parallelized
    - One worker can dig a 1 m × 1 m × 1 m large hole in one hour
    - 100 workers can dig a 100 m × 1 m × 1 m wide trench in one hour

Motivation

- Parallel computing uses multiple similar components to solve problems faster
  - It can also be used to solve larger problems
  - For example, higher resolutions in climate models
- Multiple processors can process more instructions in the same time
  - More main memory or storage can be used to process larger problems
- Not all problems can be reasonably parallelized
  - One worker can dig a 1 m × 1 m × 1 m large hole in one hour
  - 100 workers can dig a 100 m × 1 m × 1 m wide trench in one hour
  - 100 workers cannot dig a 1 m × 1 m × 100 m deep hole in one hour

Motivation...

- Parallel computers consist of multiple processing units
  - Can work on problems concurrently
- Processing unit can describe different resources
  - Arithmetic Logical Unit (ALU)
  - Floating Point Units (FPU)

Motivation…

- Parallel computers consist of multiple processing units
    - Can work on problems concurrently
- Processing unit can describe different resources
    - Arithmetic Logical Unit (ALU)
    - Floating Point Units (FPU)
    - Cores
    - Processors

Motivation...

- Parallel computers consist of multiple processing units
  - Can work on problems concurrently
- Processing unit can describe different resources
  - Arithmetic Logical Unit (ALU)
  - Floating Point Units (FPU)
  - Cores
  - Processors
  - Computers

Motivation...

- Parallel computers consist of multiple processing units
  - Can work on problems concurrently
- Processing unit can describe different resources
  - Arithmetic Logical Unit (ALU)
  - Floating Point Units (FPU)
  - Cores
  - Processors
  - Computers
  - Clusters

Motivation…

- Until ca. 2005: Performance increase via clock rate
  - Doubling the clock rate will usually double application performance
  - It is also necessary to increase memory bandwidth etc.
- Since ca. 2005: Performance increase via core count
  - Clock rate cannot be increased further due to power consumption/heat
  - Power increases linearly with frequency (and quadratically with voltage)
  - The biggest supercomputers have more than 10,000,000 cores

Motivation…

- “If you were plowing a field, which would you rather use: two strong oxen or 1024 chickens?” – Seymour Cray
    - If the processing units are too weak, parallelization might not be worth it

Motivation…

- “If you were plowing a field, which would you rather use: two strong oxen or 1024 chickens?” – Seymour Cray
    - If the processing units are too weak, parallelization might not be worth it
- “To pull a bigger wagon, it is easier to add more oxen than to grow a gigantic ox.” [Gropp et al., 2014]
    - It is much easier to scale a system by adding more processing units
    - See previous slide, it is not easily possible to increase clock rates further

Quiz

- What would you rather use to run your parallel application?
    1. 1,024 × Raspberry Pi 4 (4,096 cores, 2,048 GB RAM, ca. € 40,000)
    2. 32 × AMD EPYC 7282 (512 cores, 2,048 GB RAM, ca. € 50,000)

Quiz

- What would you rather use to run your parallel application?
  1. 1,024 × Raspberry Pi 4 (4,096 cores, 2,048 GB RAM, ca. € 40,000) ≈ 13 TFLOPS
  2. 32 × AMD EPYC 7282 (512 cores, 2,048 GB RAM, ca. € 50,000) ≈ 27 TFLOPS

Outline

Hardware Architectures

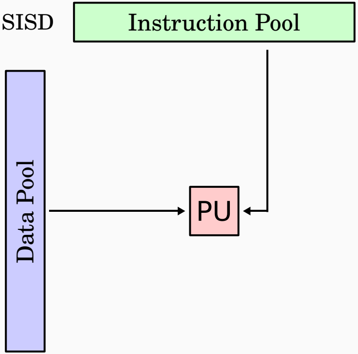
- Review
- Introduction
- Classification
- Memory Access Models
- Network
- Summary

Flynn’s Taxonomy

- Proposed by Michael J. Flynn in 1966
  - Based on the number of instruction and data streams
- Four classes
  1. SISD: Single instruction stream, single data stream
  2. SIMD: Single instruction stream, multiple data streams
  3. MISD: Multiple instruction streams, single data stream
  4. MIMD: Multiple instruction streams, multiple data streams

Flynn's Taxonomy…

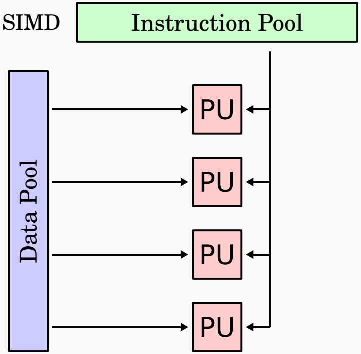
- SISD: Single processing unit
    - Executes single instruction stream
    - Instruction operates on single data stream
    - Traditional von Neumann architecture
- Can use pipelining and be superscalar
    - Execute multiple instructions per cycle



[Cburnett, 2007d]

Flynn's Taxonomy…

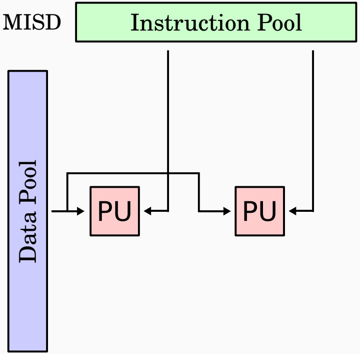
- SIMD: Single processing unit
  - Executes single instruction stream
  - Instruction operates on multiple data streams
  - Data level parallelism but not concurrency
  - Vector computers and single-core desktop processors (MMX, SSE, AVX)
- Requires vectorization
  - Not all algorithms can be vectorized
  - Often has to be done manually



[Cburnett, 2007c]

Flynn's Taxonomy…

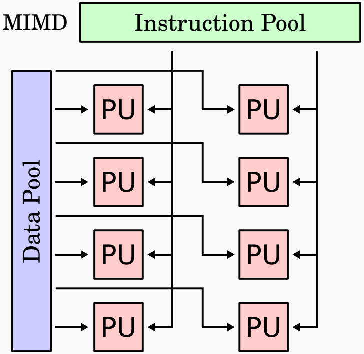
- MISD: Multiple processing units
  - Executes multiple instruction streams
  - Instructions operate on single data stream
  - Very specialized and uncommon
  - Can be used for fault tolerance
- Used in space shuttle computers
  - Calculations are performed multiple times
  - Processing units have to agree on result



[Cburnett, 2007b]

Flynn's Taxonomy...

- MIMD: Multiple processing units
  - Executes multiple instruction streams
  - Instructions operate on multiple data streams
  - Concurrency and data level parallelism
  - Multi-core desktop processors (AVX etc.)
- Also requires vectorization for full performance
  - Applications can still run concurrently



[Cburnett, 2007a]

Quiz

- Which architecture would you use for your new cluster?
  - SISD: Single instruction stream, single data stream
  - SIMD: Single instruction stream, multiple data streams
  - MISD: Multiple instruction streams, single data stream
  - MIMD: Multiple instruction streams, multiple data streams

Memory Access Models

- Applications running on MIMD often fall into two categories
  - SPMD: Single program, multiple data streams
  - MPMD: Multiple programs, multiple data streams
- Computers consist of multiple processors that communicate
  - Requires some kind of communication network
- Important classification: Memory access model
  - Shared and distributed memory
  - In reality, typically hybrid systems

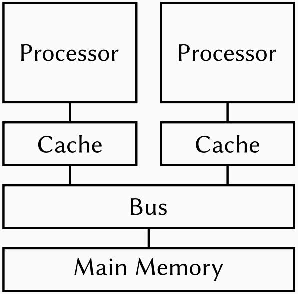
Outline

Hardware Architectures

- Review
- Introduction
- Classification
- Memory Access Models
- Network
- Summary

Shared Memory

- All processors have access to shared memory
  - There might be speed differences due to NUMA
- Typically refers to single machines
  - Can also be virtual for convenience
- Processors consist of multiple cores
  - Each core has its own caches
  - Shared cache for the whole processor
  - This is a typical MIMD architecture
- Access to shared memory via a bus
  - This also limits scalability of shared memory



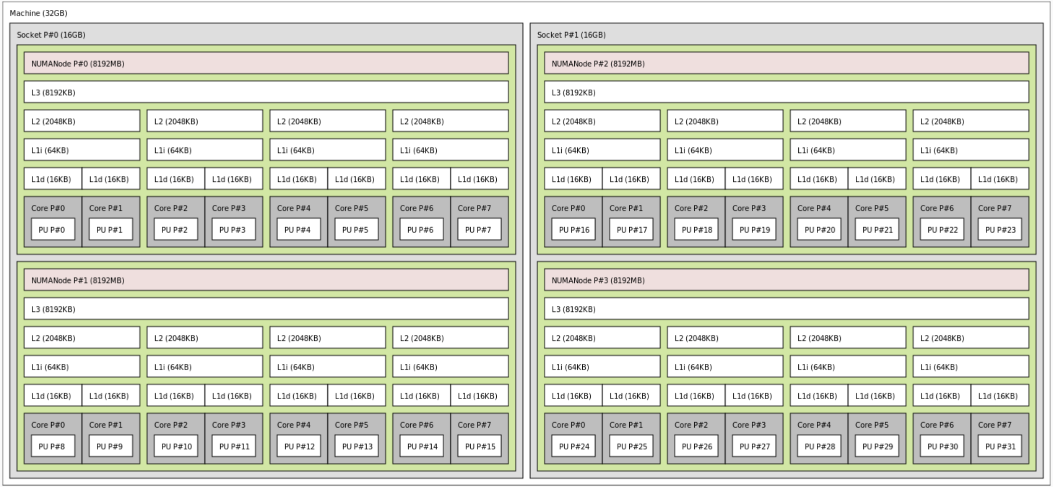
Shared Memory…

- Example: Numerical application with large matrix
  - Matrix is stored in shared main memory
  - All threads have direct access to the matrix
  - Access requires synchronization to prevent errors
  - Overall, relatively easy to use
- SPMD can be realized using separate processes
  - Differences are more theoretical in nature

Non-Uniform Memory Access

- Non-uniform memory access (NUMA) has different characteristics
    - Access to local memory with low latency and high bandwidth
    - Access to remote memory is slower
- Often the case when main memory is associated with a socket
    - Processors on the respective socket have faster access
- Processors are generally faster than their memory
    - Have to wait for main memory and thus use caches
- Global memory could slow down all processors
    - NUMA confines this problem to a single processor

Non-Uniform Memory Access…



Machine (32GB)

Socket P#0 (16GB)

NUMANode P#0 (8192MB)
L3 (8192KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1i (64KB) L1i (64KB) L1i (64KB) L1i (64KB)
L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB)
Core P#0 PU P#0 | Core P#1 PU P#1 | Core P#2 PU P#2 | Core P#3 PU P#3 | Core P#4 PU P#4 | Core P#5 PU P#5 | Core P#6 PU P#6 | Core P#7 PU P#7

NUMANode P#1 (8192MB)
L3 (8192KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1i (64KB) L1i (64KB) L1i (64KB) L1i (64KB)
L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB)
Core P#0 PU P#8 | Core P#1 PU P#9 | Core P#2 PU P#10 | Core P#3 PU P#11 | Core P#4 PU P#12 | Core P#5 PU P#13 | Core P#6 PU P#14 | Core P#7 PU P#15

Socket P#1 (16GB)

NUMANode P#2 (8192MB)
L3 (8192KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1i (64KB) L1i (64KB) L1i (64KB) L1i (64KB)
L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB)
Core P#0 PU P#16 | Core P#1 PU P#17 | Core P#2 PU P#18 | Core P#3 PU P#19 | Core P#4 PU P#20 | Core P#5 PU P#21 | Core P#6 PU P#22 | Core P#7 PU P#23

NUMANode P#3 (8192MB)
L3 (8192KB)
L2 (2048KB) L2 (2048KB) L2 (2048KB) L2 (2048KB)
L1i (64KB) L1i (64KB) L1i (64KB) L1i (64KB)
L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB) L1d (16KB)
Core P#0 PU P#24 | Core P#1 PU P#25 | Core P#2 PU P#26 | Core P#3 PU P#27 | Core P#4 PU P#28 | Core P#5 PU P#29 | Core P#6 PU P#30 | Core P#7 PU P#31

[The Portable Hardware Locality Project, 2012]

Non-Uniform Memory Access…

- Usually used as cache-coherent NUMA (ccNUMA)
  - Cache coherence ensures that all processors will see the same data
- Processors can still access all memory without cache coherence
  - Same data could be modified by different processors
  - Could lead to multiple different copies being cached
- Cache coherence can introduce significant overhead
  - Programming non-cache-coherent systems is typically too complex
  - Applications and operating systems can try to reduce overhead

Non-Uniform Memory Access…

- NUMA makes it necessary to take better care of memory allocation
  - Important to only access local memory as much as possible
  - Can be influenced by defining processor and memory affinity
- numad automatically monitors memory usage
  - Will migrate memory between NUMA nodes if necessary
  - Usually not available or enabled by default
- `numactl` can be used to specify affinity manually
- `lscpu` will also show some basic information

```
$ lscpu | grep NUMA
NUMA node(s):                     1
NUMA node0 CPU(s):                0-11
```

False Sharing

- Cache coherence may cause an effect called false sharing
  - Even if two processors do not access the same data, it may appear shared
  - Cache coherence will then cause cache invalidations
- Caches use so-called cache lines of several bytes
  - For instance, accessing an `int` will cache 64 bytes
  - All accesses will be done using cache line granularity

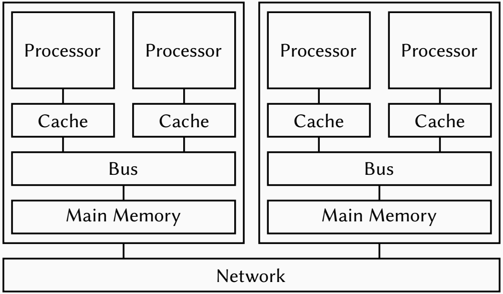
False Sharing

- Cache coherence may cause an effect called false sharing
  - Even if two processors do not access the same data, it may appear shared
  - Cache coherence will then cause cache invalidations
- Caches use so-called cache lines of several bytes
  - For instance, accessing an `int` will cache 64 bytes
  - All accesses will be done using cache line granularity
- Example: Two threads accessing different `struct` members
  - Thread 0 accesses x, thread 1 accesses y

Thread 0	$\cdots$	$\underline{x}$	y	$\cdots$
Thread 1	$\cdots$	x	$\underline{y}$	$\cdots$
	28 B	4 B	4 B	28 B

Distributed Memory

- Processors only have access to own memory
  - Typically with shared memory architecture
- Usually refers to a cluster of machines
  - Could theoretically be used inside machine
- Machines are connected via a network
  - Determines scalability and performance
  - Different network technologies and topologies



Distributed Memory…

- Example: Numerical application with large matrix
  - Matrix has to be cut into pieces and be distributed
  - Processes only have access to their local part
  - Requires communication and synchronization
  - Required data has to be sent around
  - Overall, harder to use
- Typical application of SPMD
  - A single MPI application is run on multiple machines

Comparison

- Distributed memory can be scaled almost arbitrarily
    - The largest machines have up to 10,000,000 cores in several thousand nodes
    - Requires sophisticated network topologies due to large number of nodes
    - Programming is relatively complex because of message passing
- Shared memory has limited scalability
    - Machines usually have two to four processors with a few dozen cores
    - Usage and programming is much easier due to compiler-based approaches
    - Still requires sophisticated programming to achieve very good performance

Comparison…

- Distributed memory
    - Scalability: Up to 10,000,000 cores
    - Programming: Communication and synchronization via message passing
    - Address space: Many independent spaces
- Shared Memory
    - Scalable: Up to 128 cores
    - Programming: Communication and synchronization via shared variables, locks etc.
    - Address space: One shared space
- Can we somehow get the benefits of both?

Distributed Shared Memory

- There is also distributed shared memory, which combines both approaches
  - Multiple machines with distributed memory look like shared memory
  - Partitioning tries to exploit locality to increase performance
- Can be provided by the operating system or a library
  - Operating system approach hides distributed memory from the user
- Partitioned global address space (PGAS) is a related programming model
  - Available in Unified Parallel C, Co-Array Fortran, Chapel etc.

```
use CyclicDist;
config const n = 100;
forall i in {1..n} dmapped Cyclic(startIdx=1) do
    writeln("Hello from iteration ", i, " of ", n, " on node ", here.id);
```

[Chapel Contributors, 2021]

Quiz

- What are potential problems with distributed shared memory?
  1. Data placement can lead to performance problems
  2. Unaware applications could try to use all cores
  3. Core migrations could destroy cache locality

Outline

- Hardware Architectures
  - Review
  - Introduction
  - Classification
  - Memory Access Models
  - Network
  - Summary

Overview

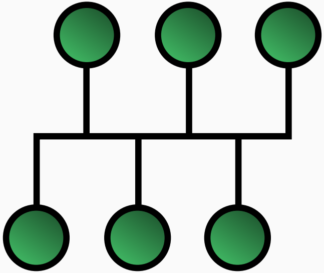
- Scalability: Ambiguous, as it can apply to different components
    - Still probably the most used metric in HPC
- How big we can make something while keeping the benefits
    - How well an application can run on more cores/nodes
    - How easy it is to increase a network's size
    - How well invested money correlates with improved performance
- Let's take a look at different network topologies and their scalability

Overview…

- Distributed memory systems have to be connected via a network
  - Different network technologies and topologies available
- Simple topologies are usually enough for smaller systems
  - Shared memory: Bus
  - Distributed memory: Star
- More complex topologies are possible
  - Fat trees, torus, dragonfly, tofu (torus fusion)
- Several important performance metrics
  - Latency, bandwidth, collisions

Bus

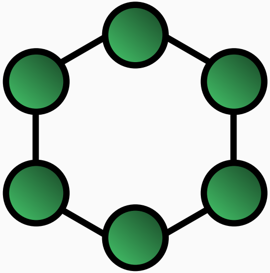
- All nodes are connected to single cable
  - Everyone receives everything and discards unnecessary data
- Simple architecture with only one cable
  - Easy to extend by making the cable longer
  - No switch required, lower chance for failures
- Scalability is problematic
  - Bandwidth is shared, limited by single cable
  - Potential for collisions



[WDGraham, 2008a]

Ring

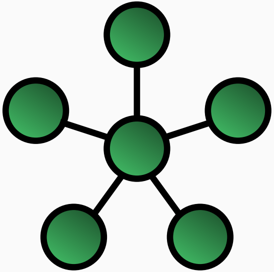
- Similar to a looped bus
  - Messages are forwarded to final recipient
- Simple architecture with multiple cables
  - Easy to extend by changing two connections
  - No switch required, no central component
- Forwarding causes delays
  - A lot of unnecessary traffic, multiple hops
- Token ring not necessarily a ring



[WDGraham, 2008c]

Star

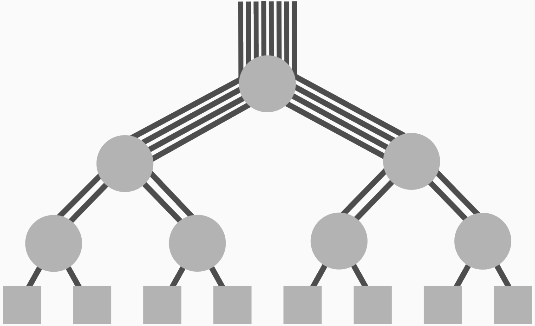
- All nodes connected to single switch
  - Only one switch required, given enough ports
- Scalable to an extent
  - Add and remove nodes without changes
  - Cable fault does not influence others
- Switch is single point of failure
  - Limited number of ports also bad for scalability



[WDGraham, 2008d]

Fat Tree

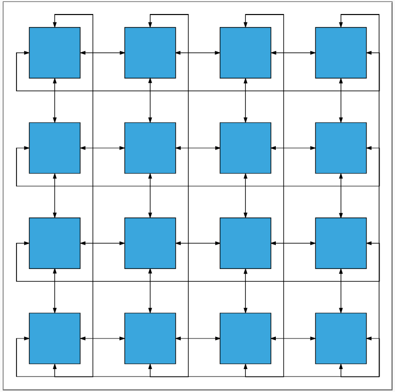
- Larger systems use hierarchical topologies
  - A fat tree has different throughputs depending on the tree level
  - Network topologies can get quite complex
- Fat trees can also have blocking factor (2:1)
  - Nodes in enclosure communicate at 100 %
  - Enclosures in rack communicate at 50 %
  - Racks communicate at 25 %



[Solnushkin, 2012]

Torus

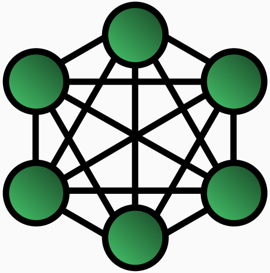
- Nodes are arranged in a mesh
  - Nodes are connected to their neighbors
  - Additional connections to opposite edges
- Possible in different dimensions
  - 1D: Two connections each (ring)
  - 2D: Four connections each (see right)
  - 3D: Six connections each
  - nD: 2n connections each
- Fewer hops due to edge connections



[Li, 2016]

Quiz

- What are disadvantages of a fully connected mesh?
  1. Communication is susceptible to hardware failures
  2. Requires special hardware for broadcasts
  3. Requires a large amount of switches
  4. Investment costs are high

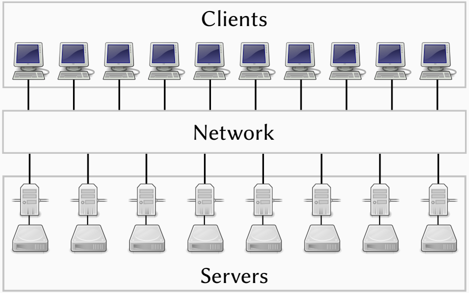


[WDGraham, 2008b]

- Supercomputers typically consist of dedicated servers
  - 1 U or blade servers for higher densities
  - Otherwise, standard server hardware
- Workstations can be aggregated into clusters
  - Allows using existing resources for complex problems
  - Typically not used for HPC due to space requirements
  - This is usually called a Beowulf cluster

Storage

- Storage is usually separated from computation
    - Computation: More cores, faster processors
    - Storage: More RAM, more storage devices
- Compute nodes have small local storage
    - Operating system, temporary files or caching
- Majority of storage in parallel file system
    - Benefit: Tuned for storage and I/O, no influence on computation
    - Drawback: Shared resource that can be impacted by others



Storage…

- File system is only used for processing, not archival
    - Hierarchical storage management and tape libraries
- Storage and I/O are often neglected
    - Not a lot of research in this area
    - Only two or three HPC-capable file systems
- Important topics for overall performance and usability
    - I/O has the potential of slowing down the whole application
    - Without storage, results cannot be written and kept for analysis

Outline

Hardware Architectures

- Review
- Introduction
- Classification
- Memory Access Models
- Network
- Summary

Summary

- Not all problems can be easily parallelized
    - Dependencies can result in a serialization of work
- Performance increases are achieved via more cores
    - Clock rates cannot be increased further due to thermal issues
- Architectures can be classified according to different characteristics
    - Instruction and data streams: SISD, SIMD, MISD and MIMD
    - Memory access model: Shared, distributed and shared distributed memory
- A wide range of network topologies are available
    - Choice depends on size of cluster, budget, reliability requirements etc.

References

[Cburnett, 2007a] Cburnett (2007a). **Flynn’s Taxonomy of a MIMD design.** https://en.wikipedia.org/wiki/File:MIMD.svg. License: CC BY-SA 3.0.

[Cburnett, 2007b] Cburnett (2007b). **Flynn’s Taxonomy of a MISD design.** https://en.wikipedia.org/wiki/File:MISD.svg. License: CC BY-SA 3.0.

[Cburnett, 2007c] Cburnett (2007c). **Flynn’s Taxonomy of a SIMD design.** https://en.wikipedia.org/wiki/File:SIMD.svg. License: CC BY-SA 3.0.

[Cburnett, 2007d] Cburnett (2007d). **Flynn’s Taxonomy of a SISD design.** https://en.wikipedia.org/wiki/File:SISD.svg. License: CC BY-SA 3.0.

[Chapel Contributors, 2021] Chapel Contributors (2021). **Chapel: Productive Parallel Programming.** https://chapel-lang.org/.

References . . .

[Gropp et al., 2014] Gropp, W. D., Lusk, E. L., and Skjellum, A. (2014). ***Using MPI - Portable Parallel Programming with the Message-Passing Interface, 3rd Edition.*** Scientific and engineering computation. MIT Press.

[Li, 2016] Li, Y. (2016). **Illustrate 2D torus.** https://en.wikipedia.org/wiki/File:2d_torus.png. License: CC BY-SA 4.0.

[Solnushkin, 2012] Solnushkin, K. S. (2012). **Fat tree varying ports.** https://clusterdesign.org/fat-trees/fat_tree_varying_ports/.

[The Portable Hardware Locality Project, 2012] The Portable Hardware Locality Project (2012). **Screenshot of lstopo/hwloc.** https://en.wikipedia.org/wiki/File:Hwloc.png. License: 3-Clause BSD.

[WDGraham, 2008a] WDGraham (2008a). **Diagram of a Bus network.** https://en.wikipedia.org/wiki/File:BusNetwork.svg.

References . . .

[WDGraham, 2008b] WDGraham (2008b). **Diagram of a fully connected Mesh network.** https://en.wikipedia.org/wiki/File:FullMeshNetwork.svg.

[WDGraham, 2008c] WDGraham (2008c). **Diagram of a Ring network.** https://en.wikipedia.org/wiki/File:RingNetwork.svg.

[WDGraham, 2008d] WDGraham (2008d). **Diagram of a Star network.** https://en.wikipedia.org/wiki/File:StarNetwork.svg.