Overview of High Performance Computing

Timothy H. Kaiser, PH.D.
tkaiser2@nrel.gov

Slides: https://github.com/timkphd/slides

Schedule

In this session we will discuss the importance of parallel and high performance computing. We will by example, show the basic concepts of parallel computing. The advantages and disadvantages of parallel computing will be discussed. We will present an overview of current and future trends in HPC hardware. We will provide a very brief overview, a comparison and contrast, of some of the paradigms of HPC , including OpenMP, Message Passing Interface (MPI) and GPU programming.

Introduction

- What is parallel computing?
- Why go parallel?
- When do you go parallel?
- What are some limits of parallel computing?
- Types of parallel computers
- Some terminology

Some links

- https://www.nrel.gov/hpc/
  - Info and link to GitHub repository with lots of examples
  - https://github.com/NREL/HPC/tree/master/slurm has versions of many of the examples I'll show today.
  - https://github.com/timkphd/examples/tree/master/openmp has the OpenMP examples
- https://nrel.github.io/HPC/

What the Exa?

```
Exa  = 1,152,921,504,606,846,976 = 2**60=1024**6 = 10**18.06
Peta =     1,125,899,906,842,624 = 2**50=1024**5 = 10**15.05
Tera =         1,099,511,627,776 = 2**40=1024**4 = 10**12.04
Giga =             1,073,741,824 = 2**30=1024**3 = 10**9.03
Mega =                 1,048,576 = 2**20=1024**2 = 10**6.02
Kilo =                     1,024 = 2**10=1024**1 = 10**3.01
```

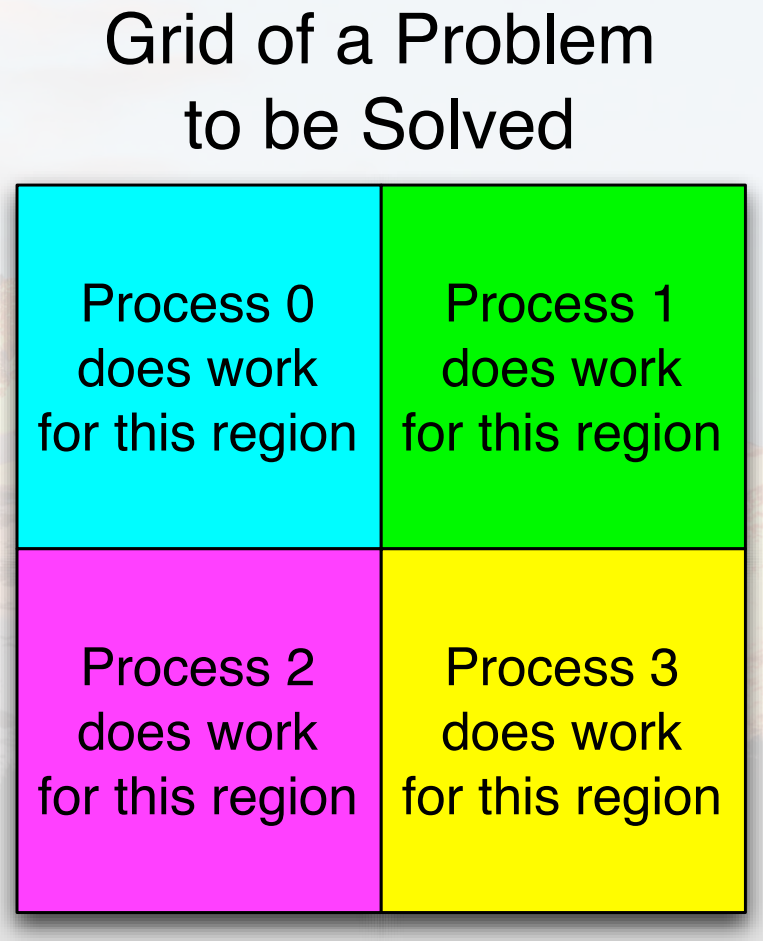
What is Parallelism?

- Consider your favorite computational application
  - One processor can give me results in N hours
  - Why not use N processors
    -- and get the results in just one hour?

The concept is simple:
Parallelism = applying multiple processors to a single problem

Parallel computing is computing by committee

- Parallel computing: the use of multiple computers or processors working together on a common task.
  - Each processor works on its section of the problem
  - Processors are allowed to exchange information with other processors



How do processors exchange information?

- Shared memory
  - Threads (mini processes) each have assess to the same memory
  - One thread can modify memory and another can see it
  - OpenMP is often used to orchestrate shared access
  - Normally restricted to a single node
- Message passing
  - An explicit message is sent from one process to another
  - MPI is the primary method for large machines
- We'll talk about OpenMP and MPI in a bit of detail

Why do parallel computing?

- Limits of single CPU computing
  - Available memory
  - Performance
- Parallel computing allows:
  - Solve problems that don't fit on a single CPU
  - Solve problems that can't be solved in a reasonable time

Why do parallel computing?

- We can run…
  - Larger problems
  - Faster
  - More cases
  - Run simulations at finer resolutions
  - Model physical phenomena more realistically

Weather Forecasting

- Atmosphere is modeled by dividing it into three-dimensional regions or cells
  - 1 mile x 1 mile x 1 mile (10 cells high)
  - about 500×10^6 cells.
- The calculations of each cell are repeated many times to model the passage of time.
- About 200 floating point operations per cell per time step or 10^{11} floating point operations necessary per time step
- 10 day forecast with 10 minute resolution => 1.5×10^{14} flop
  - 100 Mflops would take about 17 days
  - 1.7 Tflops would take 2 minutes
  - 17 Tflops would take 8 seconds
  - 105 Tflops would take 1.3 seconds

What might you want to do if running for 1.3 seconds?

Modeling Motion of Astronomical bodies (brute force)

- Each body is attracted to each other body by gravitational forces.
- Movement of each body can be predicted by calculating the total force experienced by the body.
- For N bodies, N - 1 forces / body yields N^2 calculations each time step
  - A galaxy has, 10^{11} stars => 10^9 years for one iteration
  - Using a N log N efficient approximate algorithm => about a year
  - NOTE: This is closely related to another hot topic: Protein Folding

Types of parallelism two extremes

- Data parallel
  - Each processor performs the same task on different data
  - Example - grid problems
  - Bag of Tasks or Embarrassingly Parallel is a special case
- Task parallel
  - Each processor performs a different task
  - Example - signal processing such as encoding multitrack data
  - Pipeline is a special case

Simple data parallel program

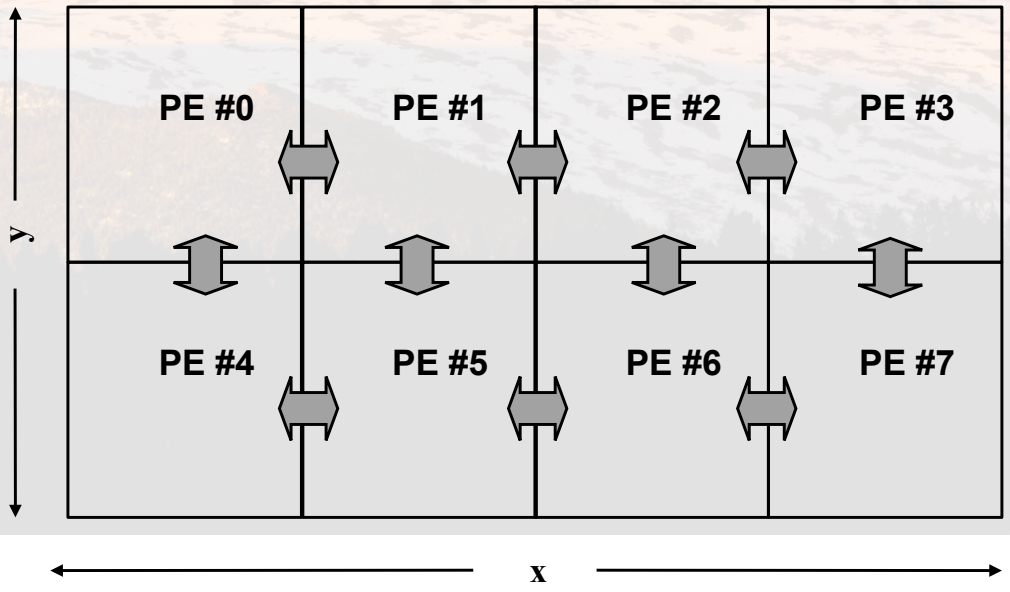
- Example: integrate 2-D propagation problem

Starting partial differential equation:

$$\frac{\partial \Psi}{\partial t} = D \times \frac{\partial^2 \Psi}{\partial x^2} + B \times \frac{\partial^2 \Psi}{\partial y^2}$$

Finite Difference Approximation:

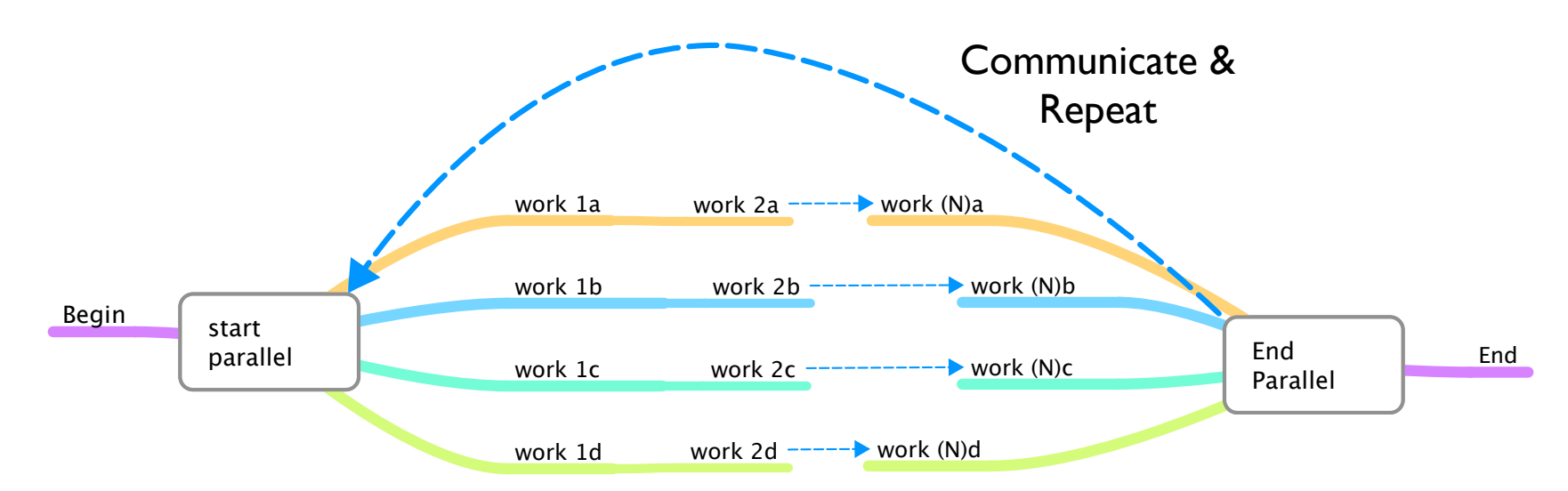
$$\frac{f_i^{n+1} - f_i^n}{\Delta t} = D \times \frac{f_{i+1,j}^n - 2f_{i,j}^n - f_{i-1,j}^n}{\Delta x^2} + B \times \frac{f_{i,j+1}^n - 2f_{i,j}^n - f_{i,j-1}^n}{\Delta y^2}$$



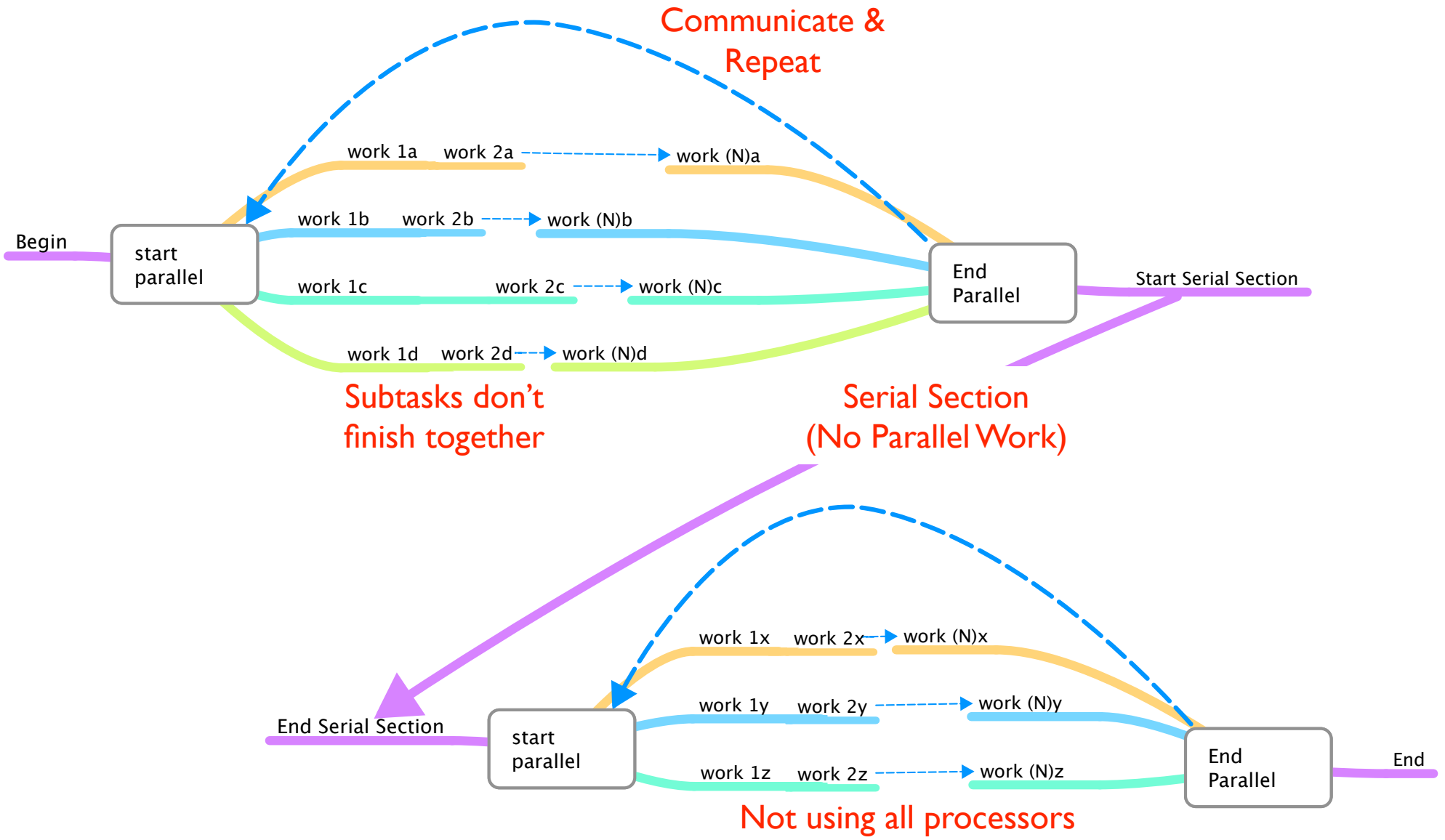
Typical Task Parallel Application

- Signal processing
  - Use one processor for each task
  - Can use more processors if one is overloaded
  - This is a pipeline

Parallel Program Structure



Parallel Problems



A "Real" example

```
#!/usr/bin/env python
from sys import argv
from os.path import isfile
from time import sleep
from math import sin,cos
#
fname="message"
my_id=int(argv[1])
print my_id, "starting program"
#
```

```
if (my_id == 0):
    myval=sin(10.0)
    notready=True
    while notready :
        if isfile(fname) :
            notready=False
            sleep(3)
            mf=open(fname,"r")
            message=float(mf.readline())
            mf.close()
            total=myval**2+message**2
        else:
            sleep(5)
    print "sin(10)**2+cos(10)**2=",total
```

```
if (my_id == 1):
    sleep(2)
    myval=cos(10.0)
    mf=open(fname,"w")
    mf.write(str(myval))
    mf.close()
```

```
print my_id, "done with program"
```

Theoretical upper limits

- All parallel programs contain:
  - Parallel sections
  - Serial sections
- Serial sections are when work is being duplicated or no useful work is being done, (waiting for others)
- Serial sections limit the parallel effectiveness
  - If you have a lot of serial computation then you will not get good speedup
  - No serial work "allows" perfect speedup
- Amdahl's Law states this formally

Amdahl's Law

- Amdahl's Law places a strict limit on the speedup that can be realized by using multiple processors.
  - Effect of multiple processors on run time

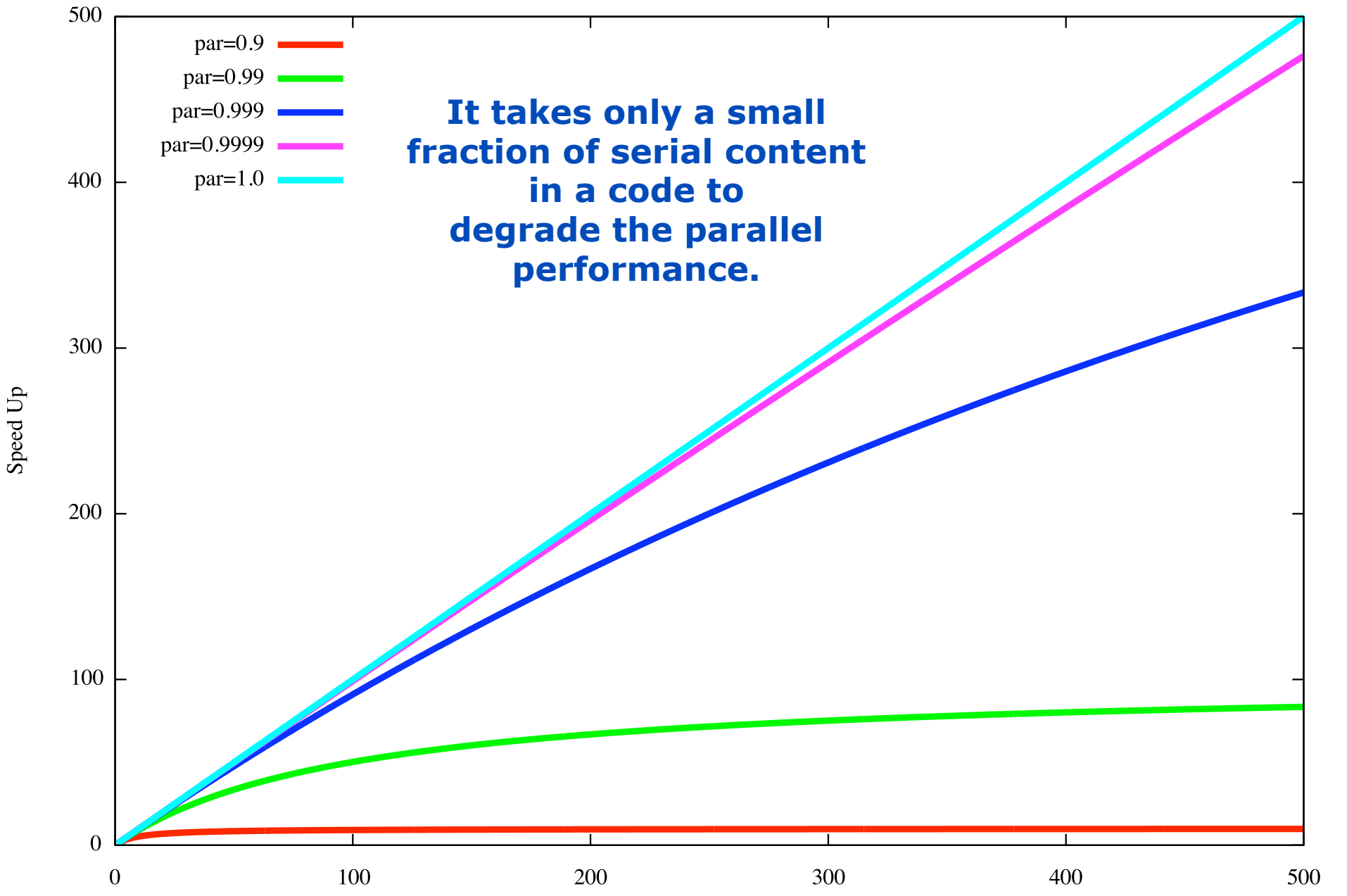
$$t_p = (f_p/N + f_s)t_s$$

  - Effect of multiple processors on speed up

$$S = t_s/t_p = \frac{1}{f_p/N + f_s}$$

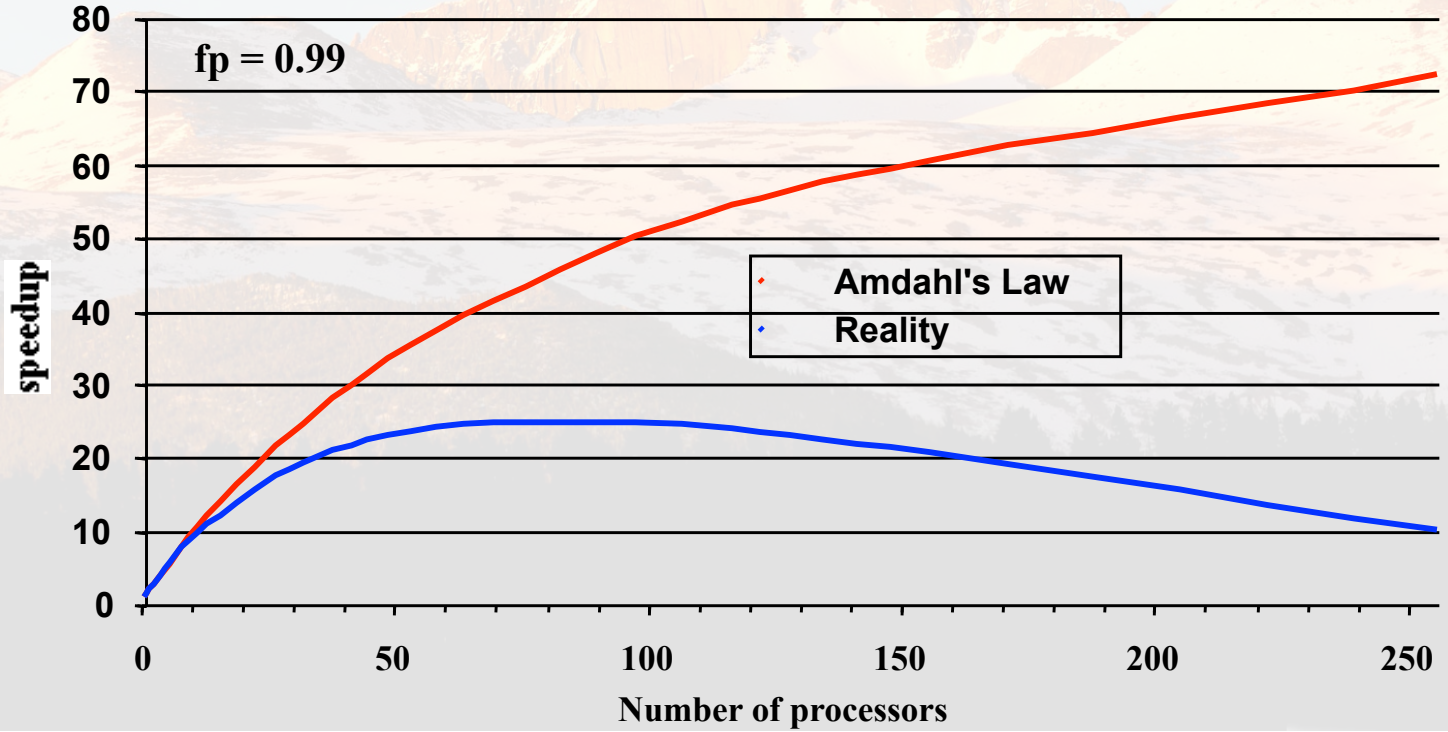
  - Where
    - Fs = serial fraction of code
    - Fp = parallel fraction of code
    - N = number of processors
  - Perfect speedup t=t1/n or S(n)=n

Illustration of Amdahl's Law

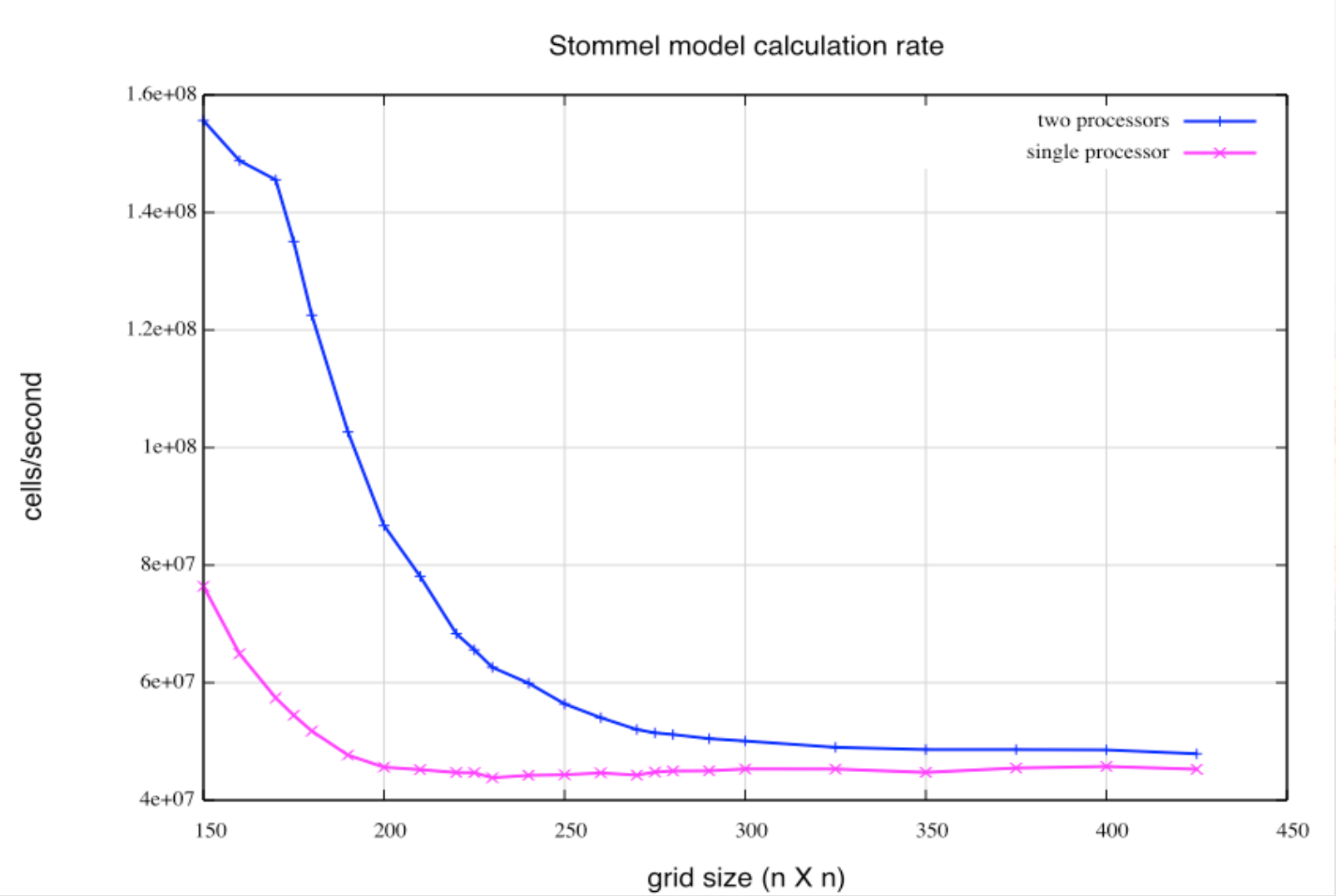


Amdahl's Law Vs. Reality

- Amdahl's Law provides a theoretical upper limit on parallel speedup assuming that there are no costs for communications.
- In reality, communications will result in a further degradation of performance



Sometimes you don't get what you expect!



Some other considerations

- Writing effective parallel application is difficult
  - Communication can limit parallel efficiency
  - Serial time can dominate
  - Load balance is important
- Is it worth your time to rewrite your application
  - Do the CPU requirements justify parallelization?
  - Will the code be used just once?

Parallelism Carries a Price Tag

- Parallel programming
  - Involves a steep learning curve
  - Is effort-intensive
- Parallel computing environments are unstable and unpredictable
  - Don't respond to many serial debugging and tuning techniques
  - May not yield the results you want, even if you invest a lot of time

Will the investment of your time be worth it?

Libraries - maybe the best path

- Libraries for many types of problems and systems
  - GPUs
  - Single node parallel
  - Multinode parallel
- People spend their life writing these
  - Going to be faster than what you write
  - No need to develop

Terms related to algorithms

- Amdahl's Law (talked about this already)
- Superlinear Speedup
- Efficiency
- Cost
- Scalability
- Problem Size
- Gustafson's Law

Superlinear Speedup

S(n) > n, may be seen on occasion, but usually this is due to using a suboptimal sequential algorithm or some unique feature of the architecture that favors the parallel formation.

One common reason for superlinear speedup is the extra cache in the multiprocessor system which can hold more of the problem data at any instant, it leads to less, relatively slow memory traffic.

Efficiency

Efficiency = Execution time using one processor over the Execution time using a number of processors

$$= \frac{t_s}{t_p \times n}$$

Its just the speedup divided by the number of processors

$$E = \frac{S(n)}{n} \times 100\%$$

Cost

The processor-time product or cost (or work) of a computation defined as
Cost = (execution time) x (total number of processors used)

The cost of a sequential computation is simply its execution time, t_s . The cost of a parallel computation is t_p x n. The parallel execution time, t_p , is given by $t_s/S(n)$

Hence, the cost of a parallel computation is given by

$$\text{Cost} = \frac{t_s n}{S(n)} = \frac{t_s}{E}$$

Cost-Optimal Parallel Algorithm
One in which the cost to solve a problem on a multiprocessor is proportional to the cost

Scalability

Used to indicate a hardware design that allows the system to be increased in size and in doing so to obtain increased performance - could be described as architecture or hardware scalability.

Scalability is also used to indicate that a parallel algorithm can accommodate increased data items with a low and bounded increase in computational steps - could be described as algorithmic scalability.

Problem size

Problem size: the number of basic steps in the best sequential algorithm for a given problem and data set size

- Intuitively, we would think of the number of data elements being processed in the algorithm as a measure of size.

- However, doubling the date set size would not necessarily double the number of computational steps. It will depend upon the problem.

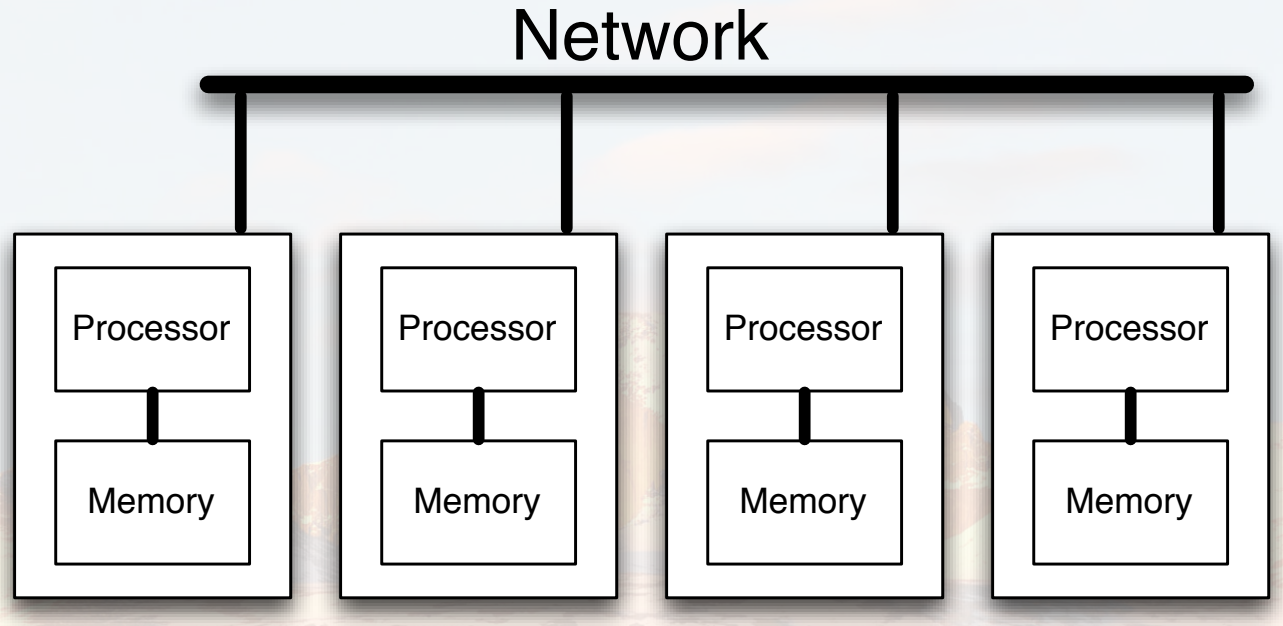
- For example, adding two matrices has this effect, but multiplying matrices quadruples operations.

Note: Bad sequential algorithms tend to scale well

Other names for Scaling

- Strong Scaling (Engineering)
  - For a fixed problem size how does the time to solution vary with the number of processors
- Weak Scaling
  - How the time to solution varies with processor count with a fixed problem size per processor

Some Classes of machines



Distributed Memory
Processors only Have access to
their local memory
"talk" to other processors over a network

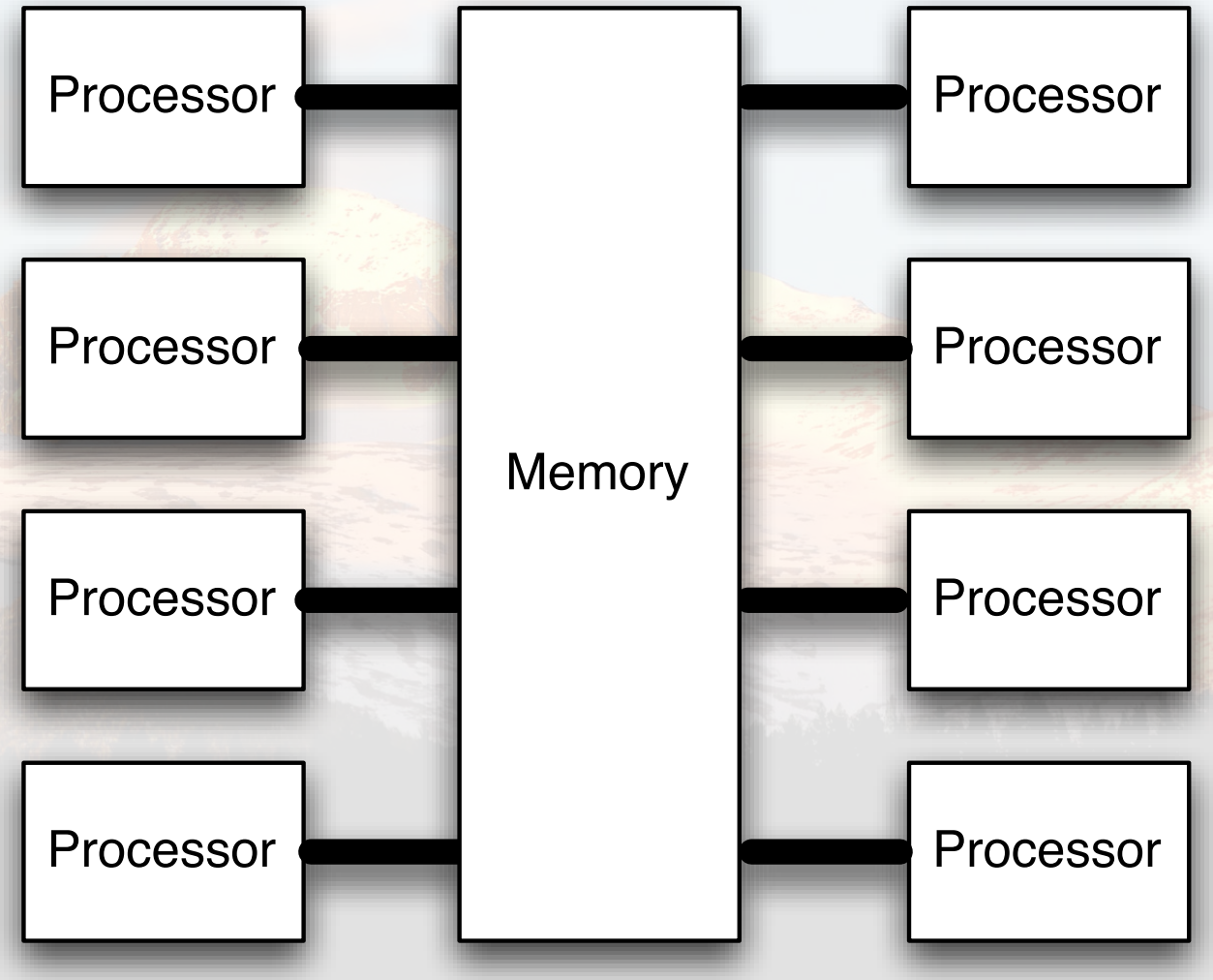
Some Classes of machines

Uniform
Shared
Memory
(UMA)

All processors
have equal access to
Memory

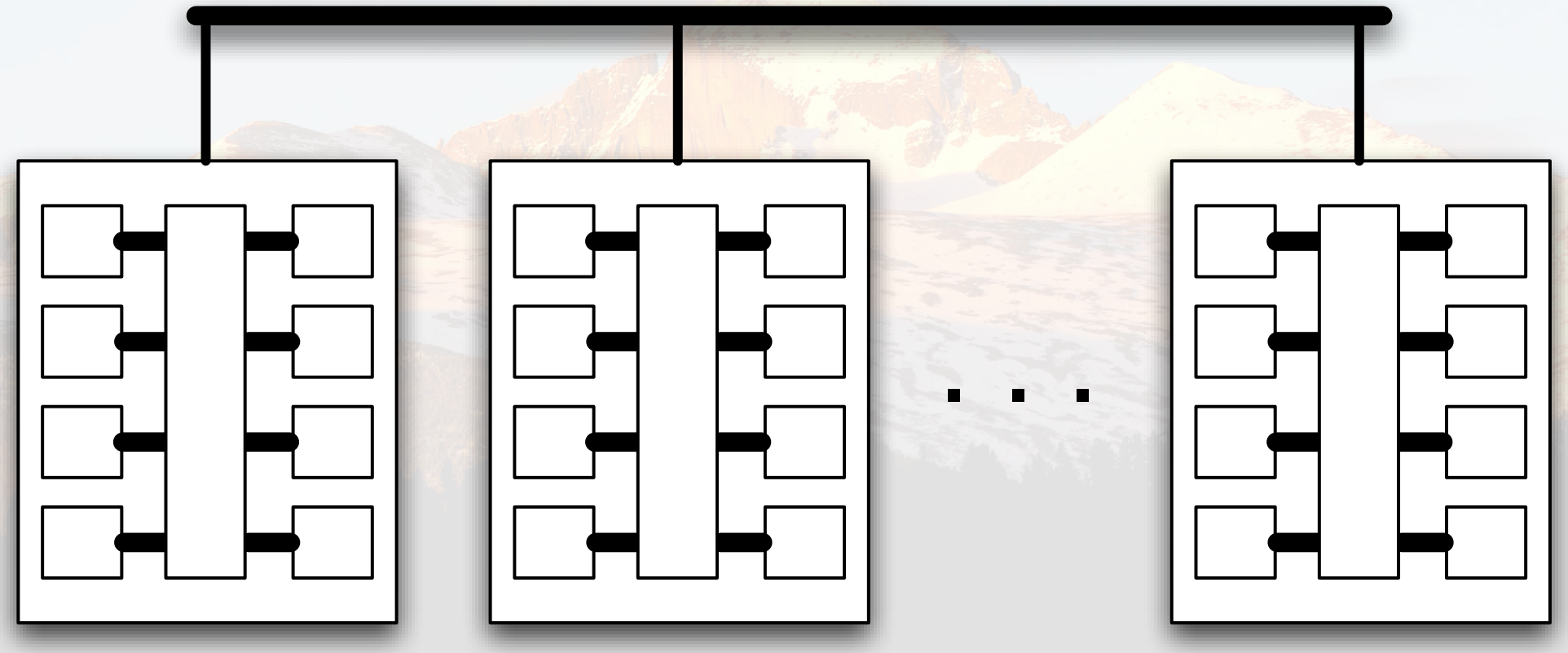
Can "talk"
via memory

This is what is normally
described as a node



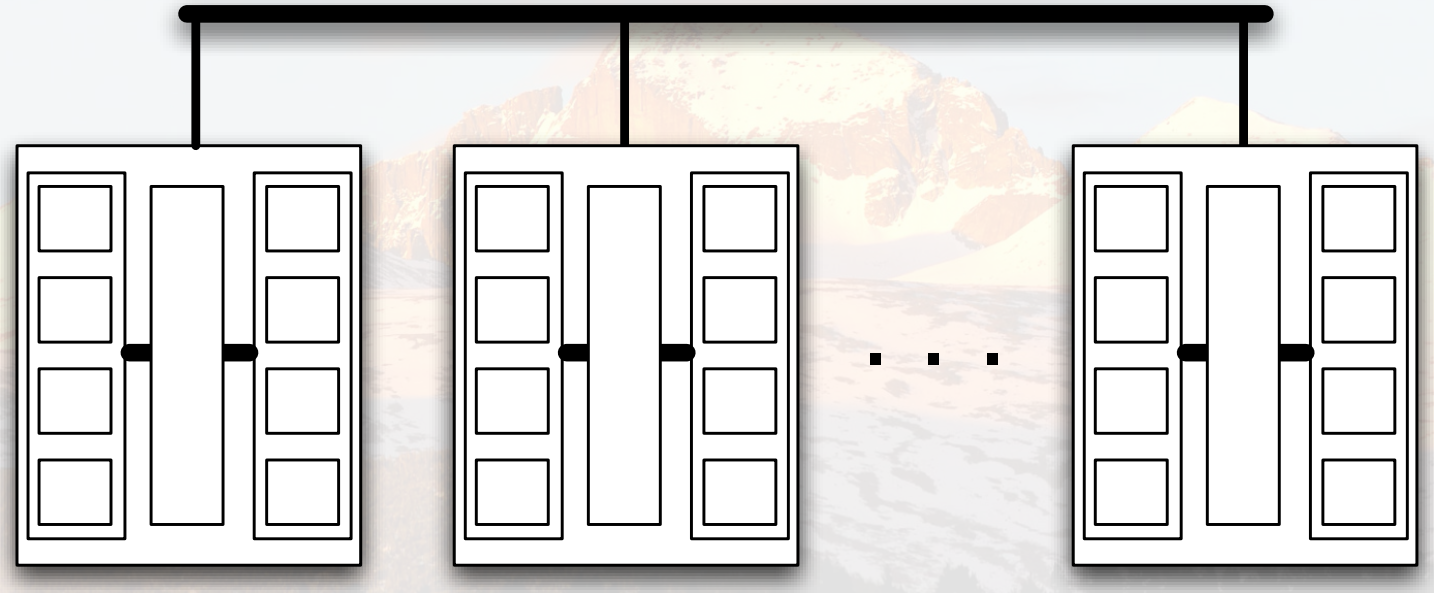
Some Classes of machines

Hybrid
Shared memory nodes
connected by a network



Some Classes of machines

More common today
Each node has a collection
of multicore chips

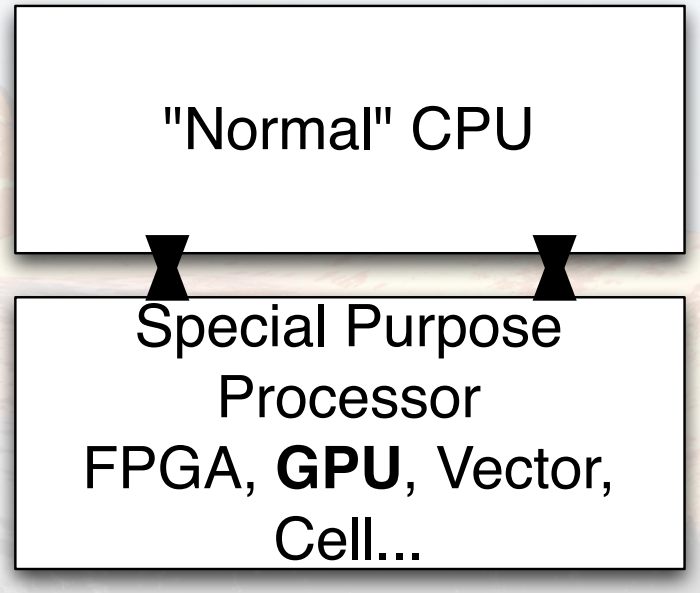


Eagle has 2600+ nodes
36 cores per node
Some nodes also have GPUS

Some Classes of machines

Accelerated Machines

- Add special purpose processors to normal processors
- Not a new concept (8086/8087) but, regaining traction
- Example: Eagle has 40+ GPU nodes with 2 GPUs each
- Speed up can be astounding!
- Issues:
  - Transfer speed between units
  - Some calculations not amenable
  - Might be hard to program



GPU vendors

- Nvidia is the most common
- AMD is also important
- ARM/Nvidia have announced CPU designed specifically to interface with GPUs
  - https://www.nvidia.com/en-us/data-center/grace-cpu/

Recent Nvidia GPUs

https://en.wikipedia.org/wiki/Ampere_(microarchitecture)

Accelerator	Architecture	FP32 CUDA Cores	FP64 Cores(excl. Tensor)	INT32 Cores	Boost Clock	Memory Clock	Memory Bus Width	Memory Bandwidth	VRAM
A100 80GB	Ampere	6912	3456	6912	1410 MHz	3.2Gbit/s HBM2	5120-bit	2039GB/sec	80GB
A100	Ampere	6912	3456	6912	1410 MHz	2.4Gbit/s HBM2	5120-bit	1555GB/sec	40GB
V100	Volta	5120	2560	5120	1530 MHz	1.75Gbit/s HBM2	4096-bit	900GB/sec	16GB/32GB
P100	Pascal	3584	1792	N/A	1480 MHz	1.4Gbit/s HBM2	4096-bit	720GB/sec	16GB

Accelerator	Single Precision	Double Precision(FP64)	INT8(non-Tensor)	INT8 Tensor	INT32	FP16	FP16 Tensor	bfloat16 Tensor	TensorFloat-32(TF32) Tensor	FP64 Tensor
A100 80GB	19.5 TFLOPs	9.7 TFLOPs	N/A	624 TOPs	19.5 TOPs	78 TFLOPs	312 TFLOPs	312 TFLOPs	156 TFLOPs	19.5 TFLOPs
A100	19.5 TFLOPs	9.7 TFLOPs	N/A	624 TOPs	19.5 TOPs	78 TFLOPs	312 TFLOPs	312 TFLOPs	156 TFLOPs	19.5 TFLOPs
V100	15.7 TFLOPs	7.8 TFLOPs	62 TOPs	N/A	15.7 TOPs	31.4 TFLOPs	125 TFLOPs	N/A	N/A	N/A
P100	10.6 TFLOPs	5.3 TFLOPs	N/A	N/A	N/A	21.2 TFLOPs	N/A	N/A	N/A	N/A

Accelerator	Interconnect	GPU	L1 Cache Size	L2 Cache Size	GPU Die Size	Transistor Count	TDP	Manufacturing Process
A100 80GB	600GB/sec	GA100	20736KB(192KBx108)	40960 KB	826mm2	54.2B	400W	TSMC 7 nm N7
A100	600GB/sec	GA100	20736KB(192KBx108)	40960 KB	826mm2	54.2B	400W	TSMC 7 nm N7
V100	300GB/sec	GV100	10240KB(128KBx80)	6144 KB	815mm2	21.1B	300W/350W	TSMC 12 nm FFN
P100	160GB/sec	GP100	1344KB(24KBx56)	4096 KB	610mm2	15.3B	300W	TSMC 16 nm FinFET+

```
A100  - May 2020
V100  - Dec 2017
P100  - Jun 2016
```

Programming GPUs

- Cuda - lowest level, closest to hardware
- Directives - ACC - like OpenMP, portable across vendors
- Libraries - many available - including for ML/AI
  - Many have cpu (OpenMP) versions also
  - Tensorflow
  - pyTourch
  - OpenCV
  - cuDF - Pandas like
  - cuPY - Numpy like

Python GPU example

There's a GPU enabled version of the standard Python Library Numpy - Cupy
We do a large 2d FFT example (10240,10240)

```
[17]: al=np.random.rand(10240,10240)
      todo=10
      s = time.time()
      for i in range(1,todo+1):
          b=np.fft.fft2(al)
          print(i)
      e = time.time()
      rate=todo/(e-s)
      print("FFTs/Second=",rate)
      ratecpu=rate

      1
      2
      3
      4
      5
      6
      7
      8
      9
      10
      FFTs/Second= 0.2340631825231919
```

```
a=cp.random.rand(10240,10240,dtype = 'float32')
from tymer import tymer
todo=500
s = time.time()
for i in range(1,todo+1):
    b=cp.fft.fft2(a)
    if i % 50 == 0:
        print(i)
e = time.time()
rate=todo/(e-s)
print("FFTs/Second=",rate)
rategpu=rate
print("GPU/CPU FFT compute ratio=",rategpu/ratecpu)

50
100
150
200
250
300
350
400
450
500
FFTs/Second= 131.00105474129168
GPU/CPU FFT compute ratio= 555.8325600541957
```

Cuda - normally C

Lowest Level programming for Nvidia GPUS

Cuda from within Python:

A definition of an elementwise kernel consists of four parts:

- an input argument list
- an output argument list
- a loop body code
- and the kernel name.

A kernel that computes a squared difference

[13]:
```
inargs='float32 x, float32 y'
outargs='float32 z'
body='z = (x - y) * (x - y)'
name='squared_diff'

squared_diff = cp.ElementwiseKernel(inargs,outargs,body,name)
```

[14]:
```
x = cp.arange(10, dtype=np.float32).reshape(2, 5)
y = cp.arange(5, dtype=np.float32)
squared_diff(x, y)
```

[14]:
```
array([[ 0.,  0.,  0.,  0.,  0.],
       [25., 25., 25., 25., 25.]], dtype=float32)
```

[15]:
```
x
```

[15]:
```
array([[0., 1., 2., 3., 4.],
       [5., 6., 7., 8., 9.]], dtype=float32)
```

[16]:
```
y
```

[16]:
```
array([0., 1., 2., 3., 4.], dtype=float32)
```

We can combine methods

- Message passing and GPU
  - Dask
  - Ray
  - rlLIB
  - MPI/GPU
  - horovod
- Message passing and OpenMP
  - Vasp (program)
  - NAMD (program)
  - PetSC (Library)
  - Scalapack (Library)

Network Topology

- For ultimate performance you may be concerned how you nodes are connected.
- Avoid communications between distant node
- For some machines it might be difficult to control or know the placement of applications

Network Terminology

- Latency
  - How long to get between nodes in the network.
- Bandwidth
  - How much data can be moved per unit time.
  - Bandwidth is limited by the number of wires and the rate at which each wire can accept data and choke points

Ring

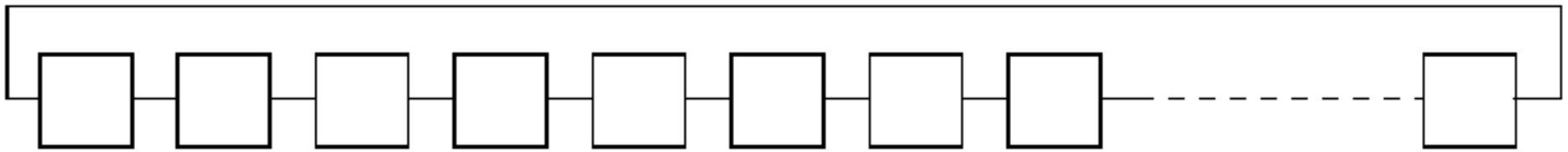


Figure 1.10 Ring.

Grid

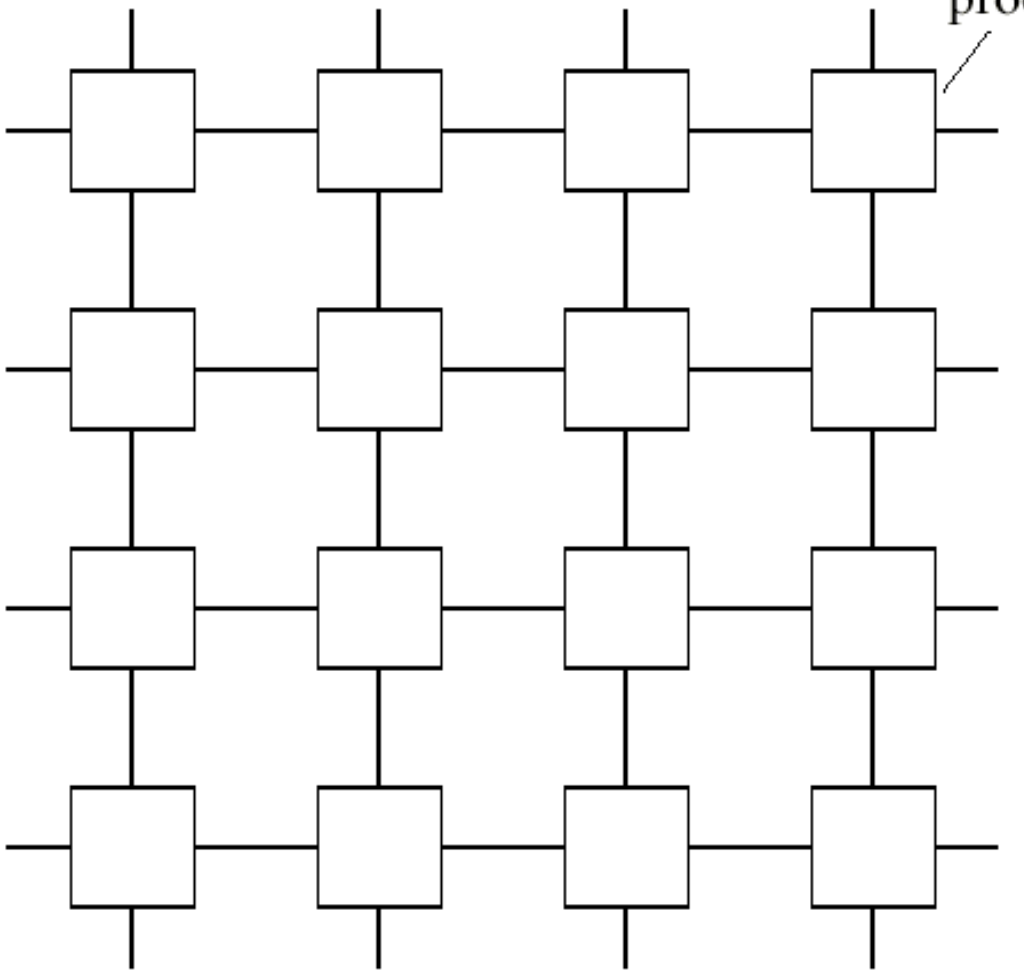
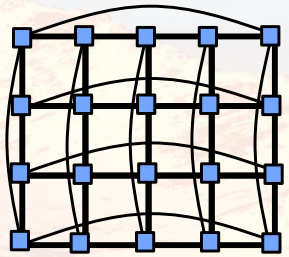


Figure 1.11 Two-dimensional array (mesh).



Wrapping produces torus

Tree

Fat tree
the lines get
wider as you
go up

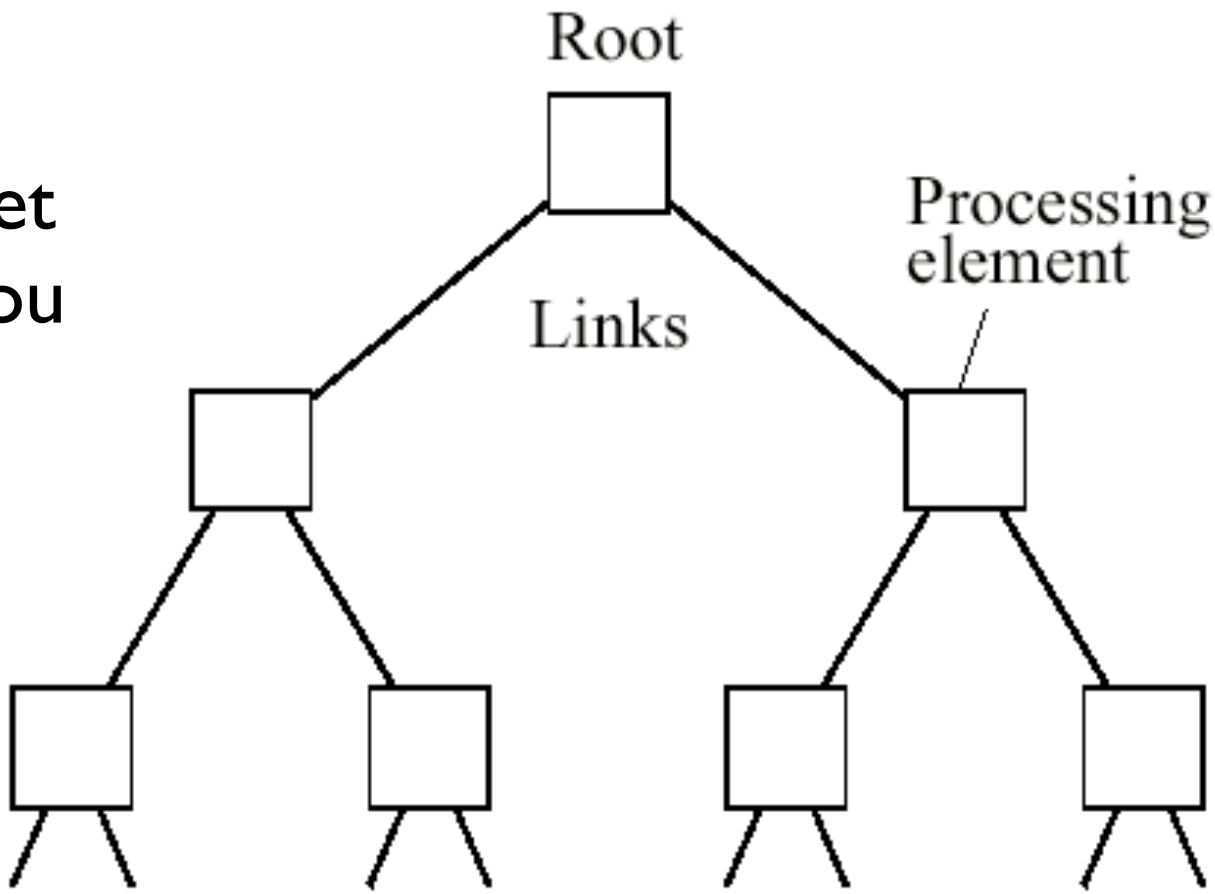
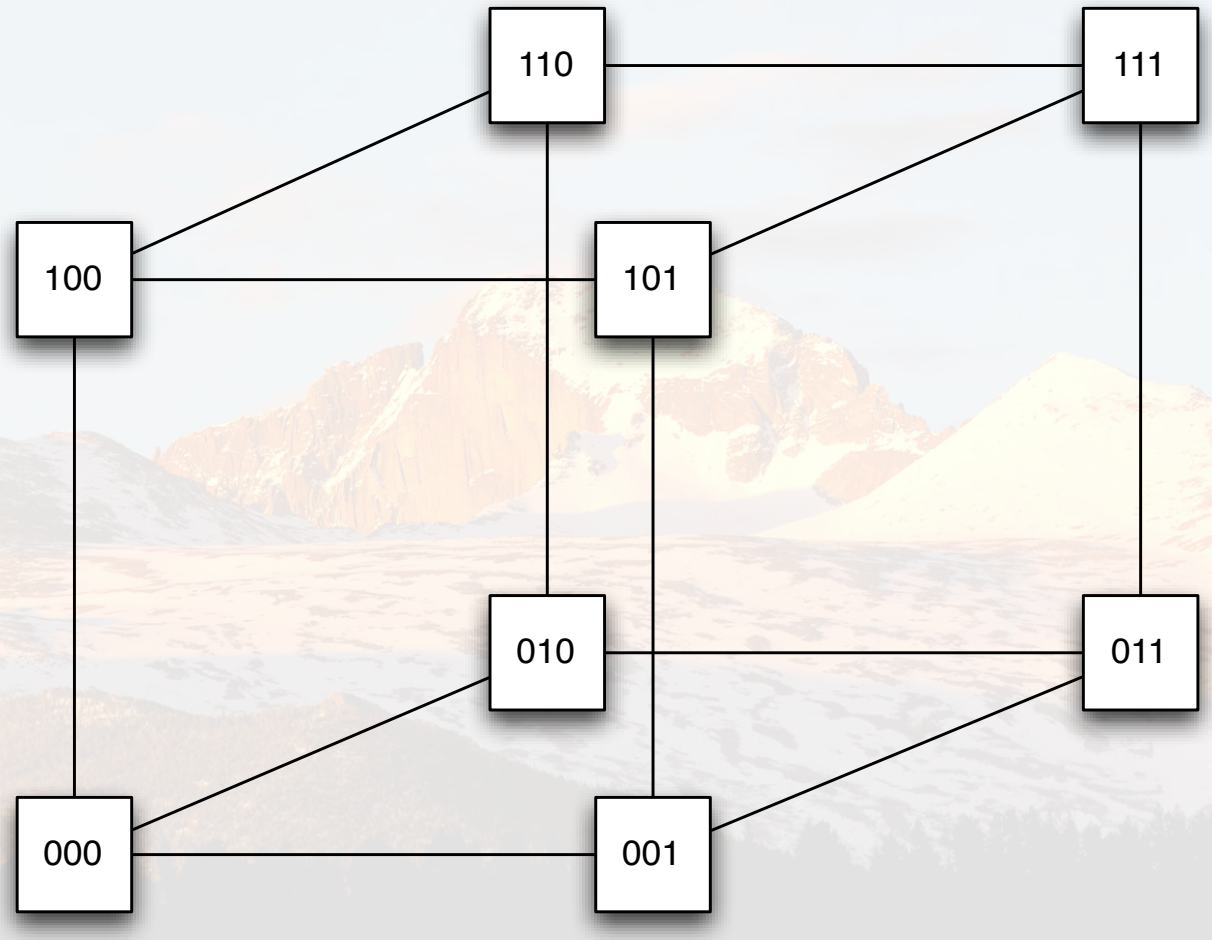


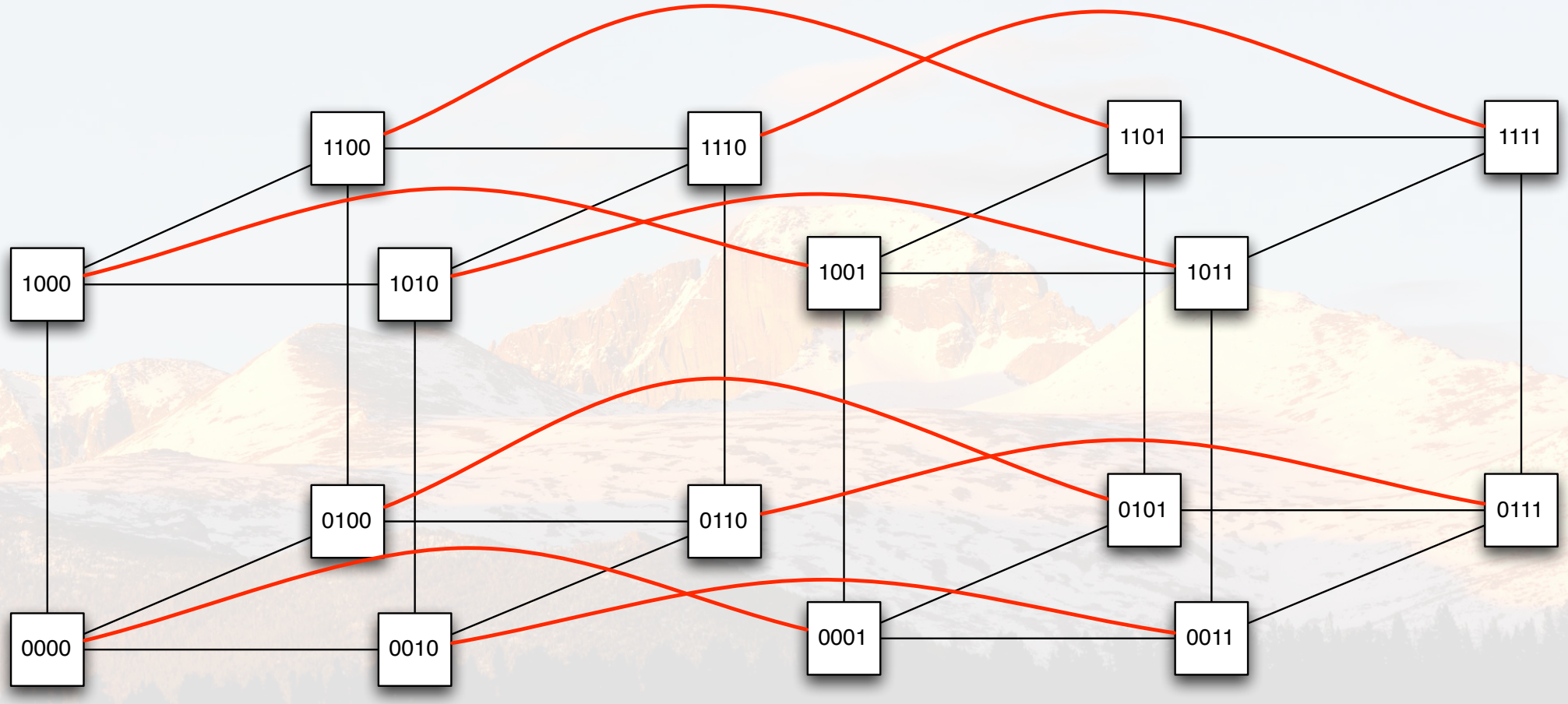
Figure 1.12 Tree structure.

Hypercube



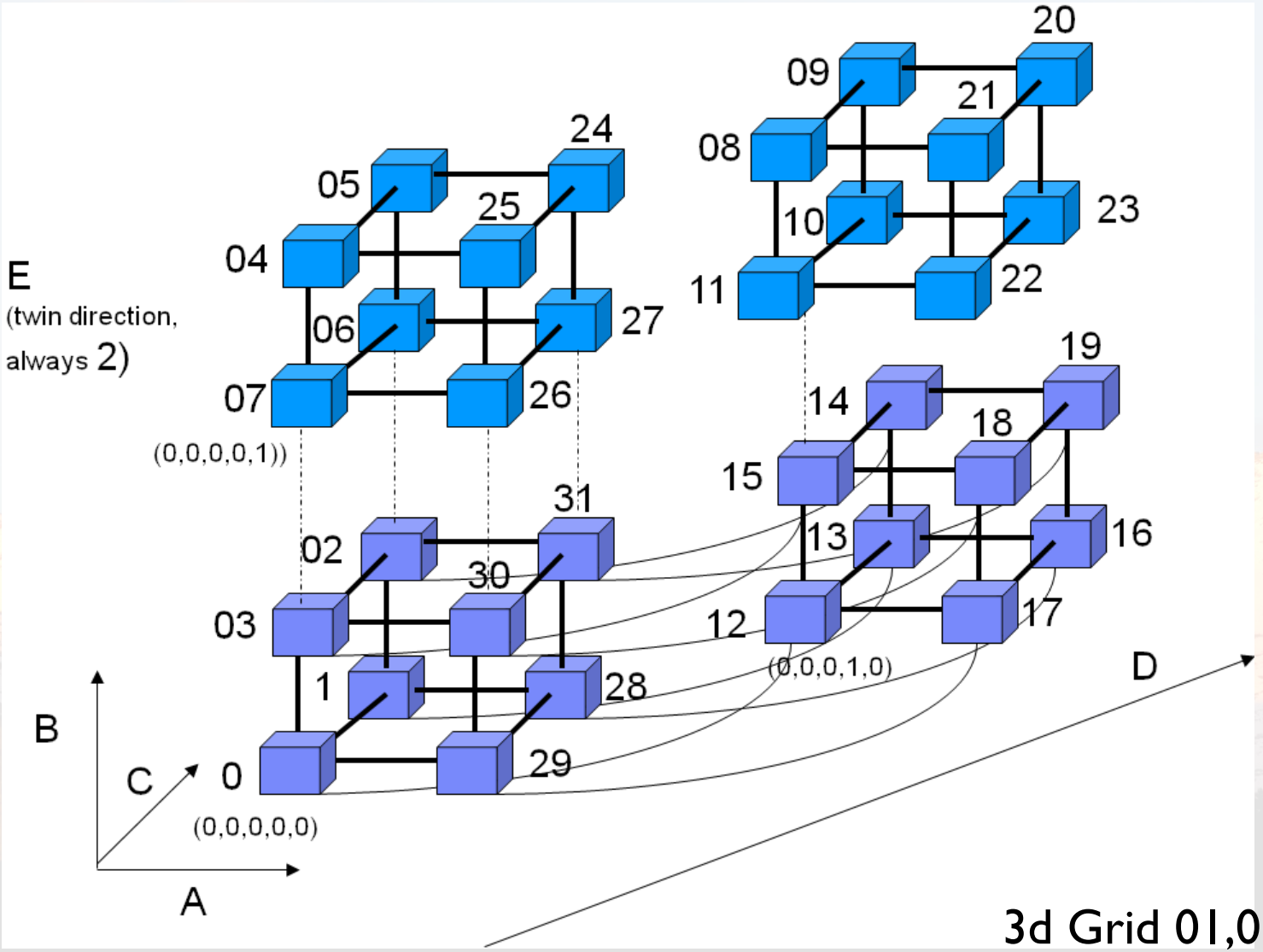
3 dimensional hypercube

4D Hypercube



Some communications algorithms are hypercube based
How big would a 7d hypercube be?

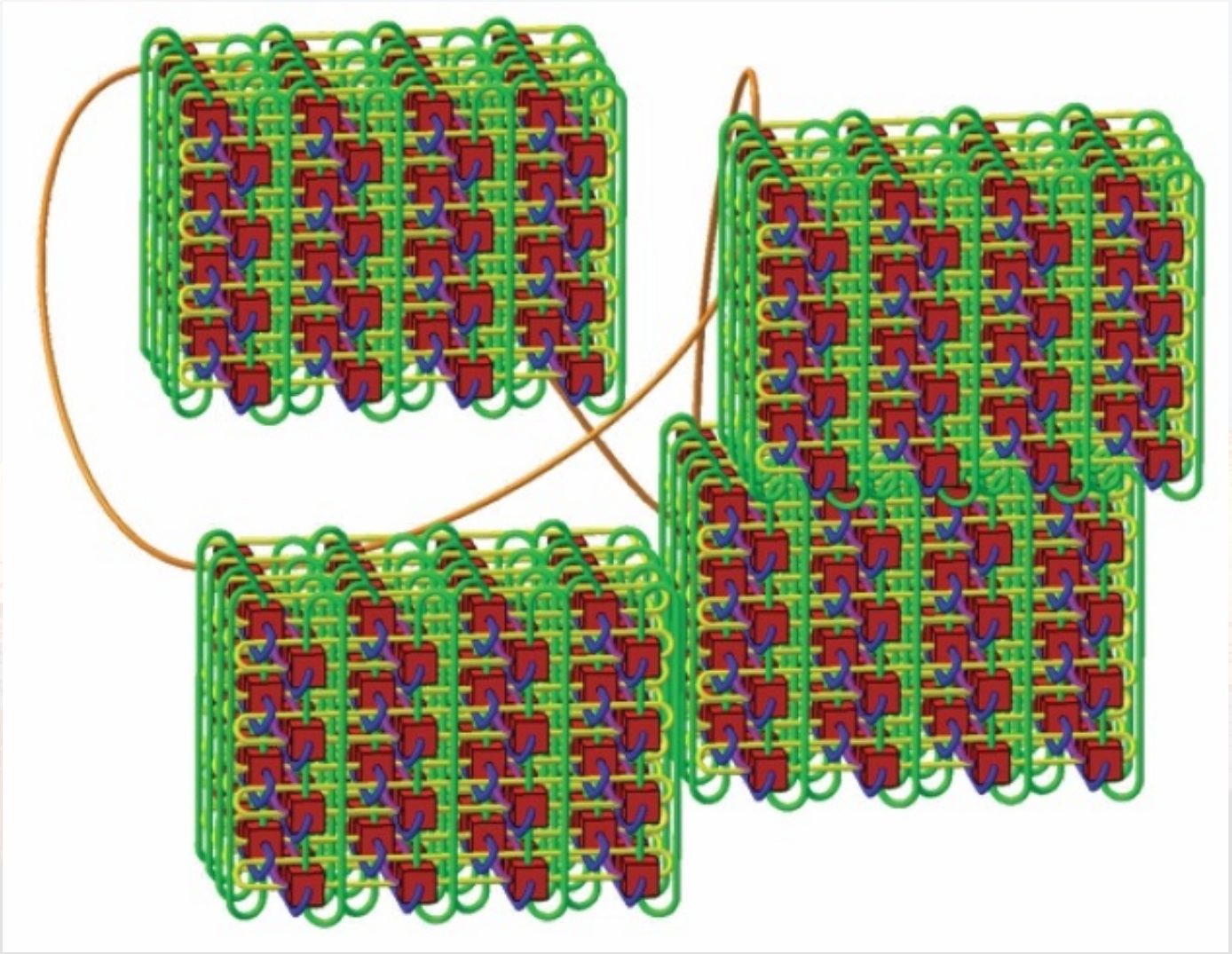
5d Torus



http://www.idris.fr/eng/turing/hw-turing-eng.html

3d Grid 01,03,29
3d Torus adds 01,03,29
5d adds 12

5d - Blue Gene Q



MidPlane - 512 nodes 4x4x4x4x2

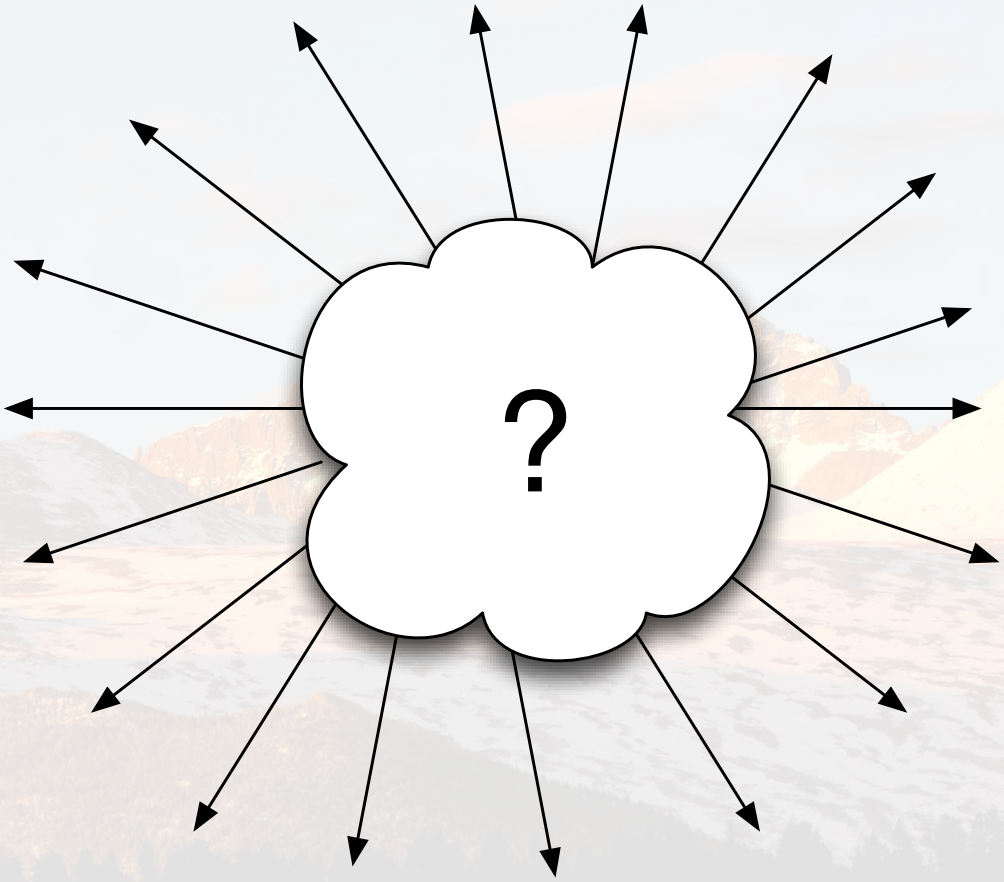
BGQ Layout

5D Torus Network

The network topology of BlueGene/Q is a five-dimensional (5D) torus, with direct links between the nearest neighbors in the ±A, ±B, ±C, ±D, and ±E directions. As such there are only a few optimum block sizes that will use the network efficiently.

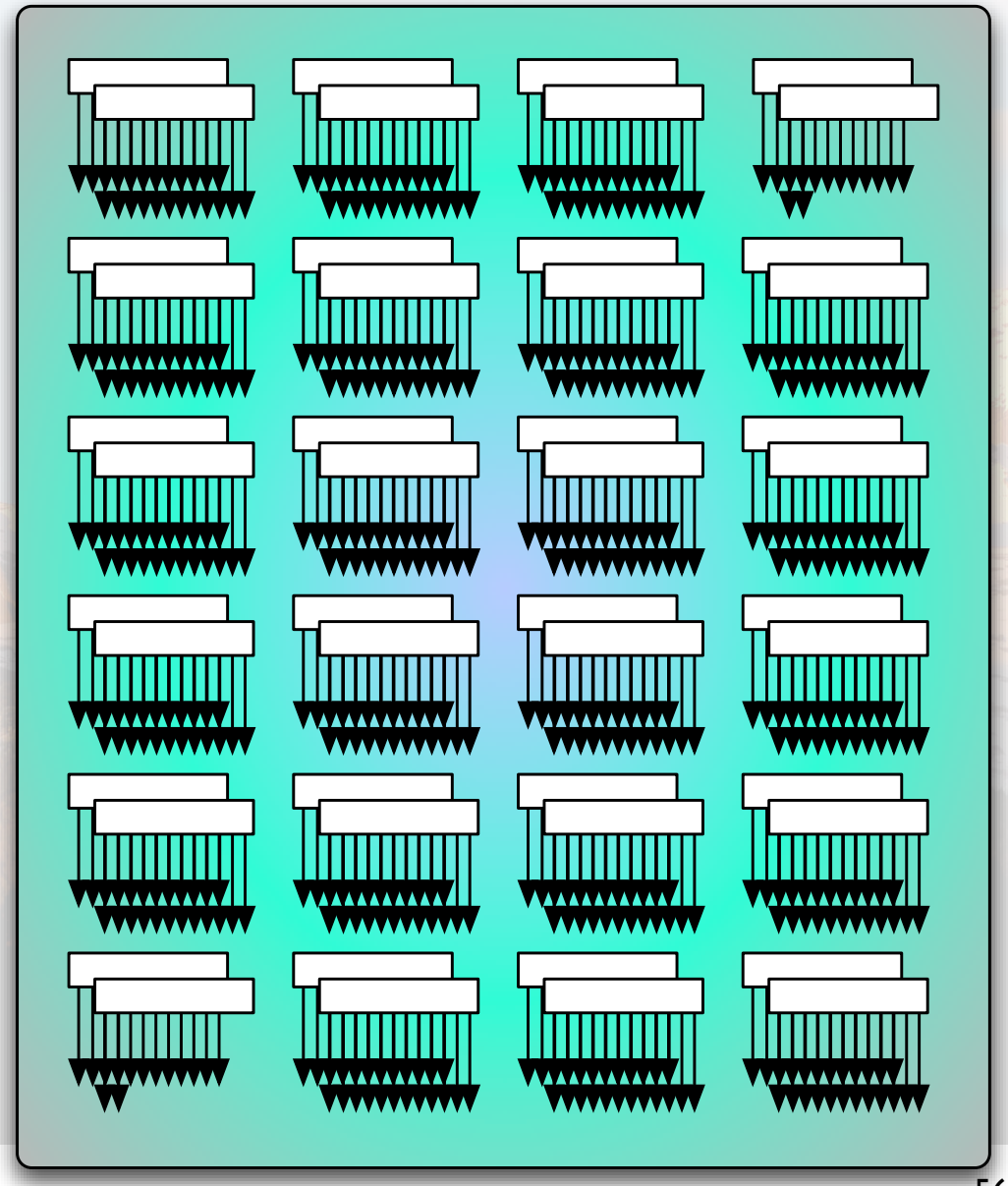
Node Boards	Compute Nodes	Cores	Torus Dimensions
1	32	512	2x2x2x2x2
2 (adjacent pairs)	64	1024	2x2x4x2x2
4 (quadrants)	128	2048	2x2x4x4x2
8 (halves)	256	4096	4x2x4x4x2
16 (midplane)	512	8192	4x4x4x4x2
32 (1 rack)	1024	16384	4x4x4x8x2
64 (2 racks)	2048	32768	4x4x8x8x2

Star



Quality depends on what is in the center

Example: An Infiniband (old) Switch



Infiniband, DDR, Cisco 7024 IB Server Switch - 48 Port

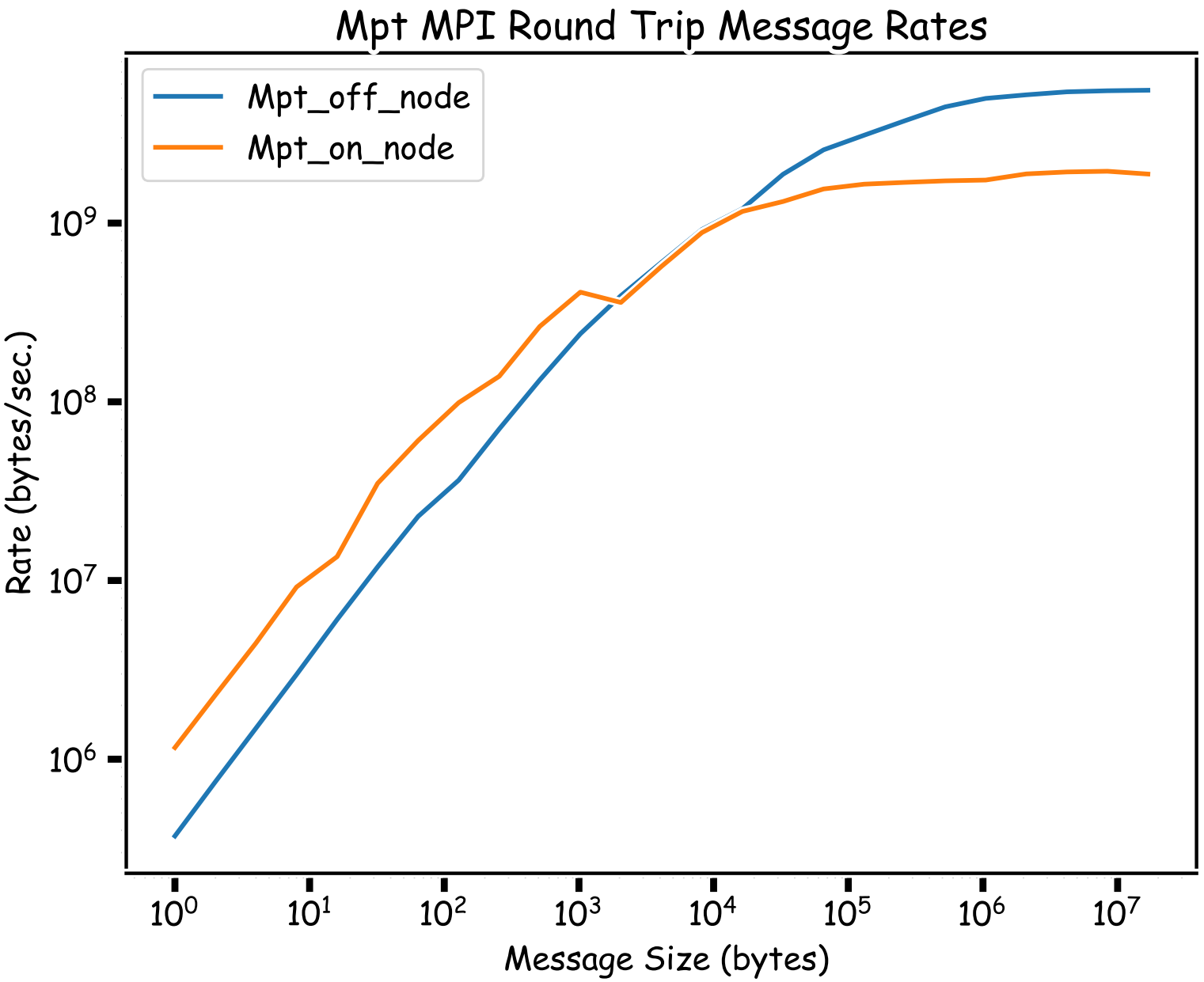
Adaptors. Each compute node has one DDR 1-Port HCA

4X DDR=> 16Gbit/sec

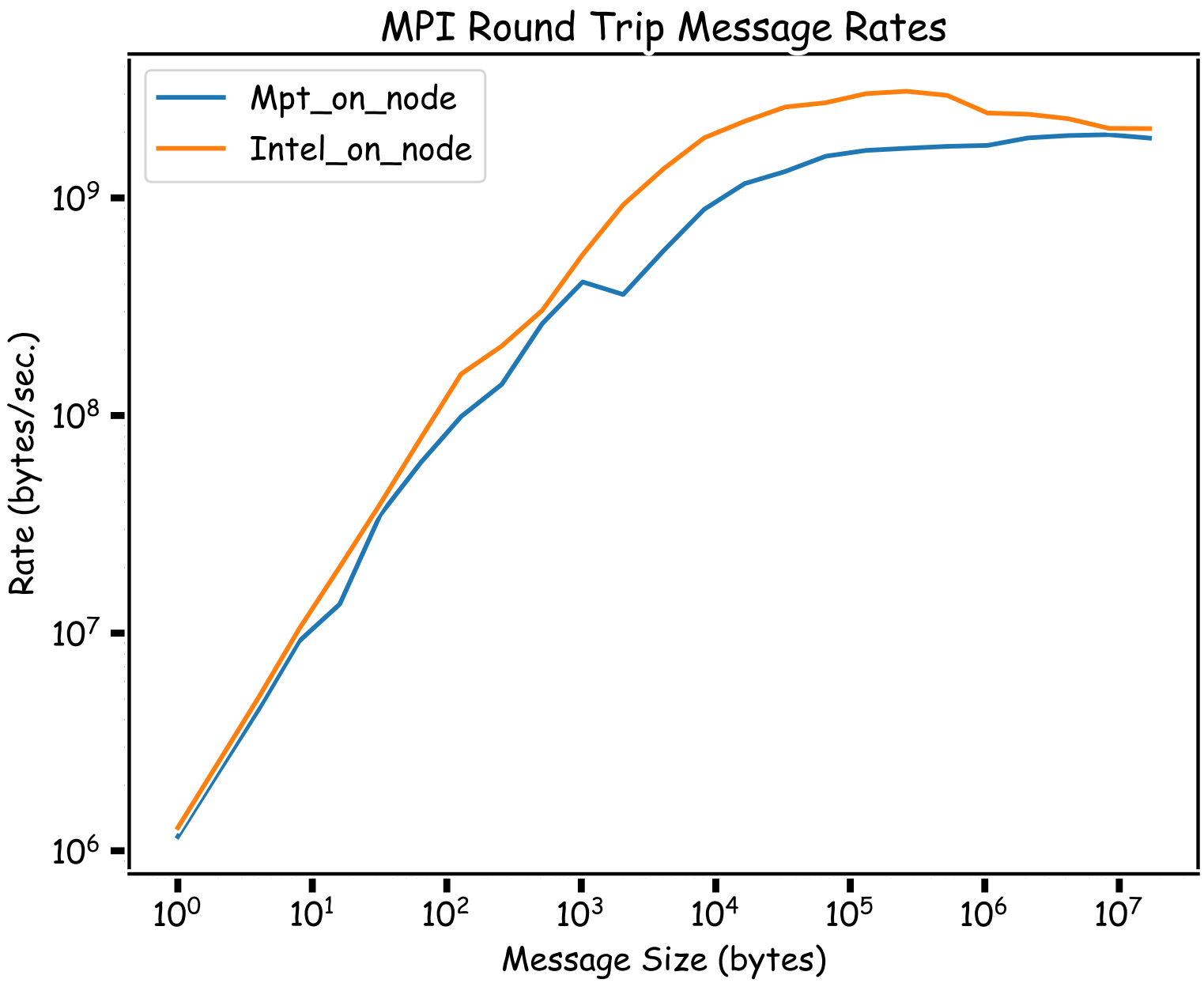
140 nanosecond hardware latency

1.26 microsecond at software level

Measured Bandwidth



Measured Bandwidth



Infiniband Versions

Characteristics

		SDR	DDR	QDR	FDR10	FDR	EDR	HDR	NDR	XDR
Signaling rate (Gbit/s)		2.5	5	10	10.3125	14.0625[7]	25.78125	50	100	250
Theoretical effective throughput (Gb/s)[8]	**for 1 link**	2	4	8	10	13.64	25	50	100	250
	for 4 links	8	16	32	40	54.54	100	200	400	1000
	for 8 links	16	32	64	80	109.08	200	400	800	2000
	for 12 links	24	48	96	120	163.64	300	600	1200	3000
Encoding (bits)		8b/10b			64b/66b				PAM4	t.b.d.
Adapter latency (µs)[9]		5	2.5	1.3	0.7	0.7	0.5	less?	t.b.d.	t.b.d.
Year[10]		2001, 2003	2005	2007	2011	2011	2014[11]	2018[11]	2021[11]	after 2023?

https://en.wikipedia.org/wiki/InfiniBand

Libraries -maybe the best path

- Libraries for many types of problems and systems
  - GPUs
  - Single node parallel
  - Multinode parallel
- People spend their life writing these
  - Going to be faster than what you write
  - No need to develop

Yes, you have seen this slide before.