# Learning High Performance Computing with a Pi Cluster

Martin Siebenborn

Universität Trier

June 11, 2016

ALGORITHMIC OPTIMIZATION

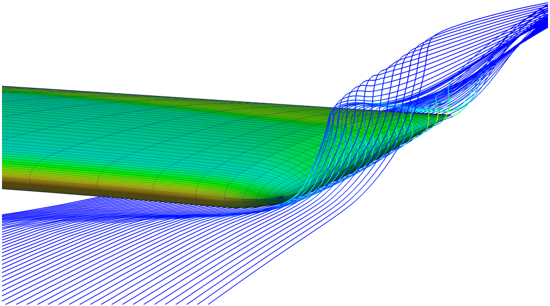
SPPEXA

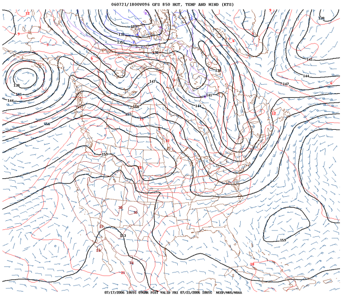
# Outline

- What is HPC?
- Constructing manual for a Pi cluster
- Educating HPC for math students
- Should I build such a cluster for my projects?

# Motivation for high performance computing

- One of the driving forces for the development of super computers are numerical simulations, e.g. fluid dynamics.





Weather prediction, Source: Wikipedia

- Physical principals + numerical methods $\rightarrow$ very large linear system
- Solve $Ax = b$ with $A \in \mathbb{R}^{n \times n}, x, b \in \mathbb{R}^n$ for $n \approx 10^9$

# Moor's law I

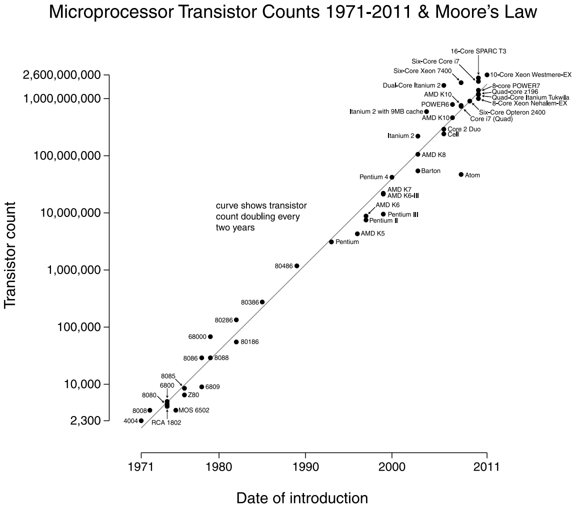
- The complexity of integrated circuits **doubles every 18 months**, e.g. measured by the number of transistors
- **But** this can not be continued arbitrarily
- Physical limits in the manufacturing of semiconductors

# Moor's law I

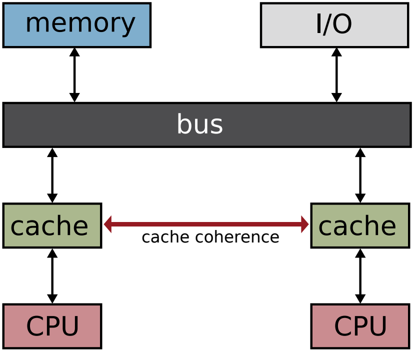
- The complexity of integrated circuits **doubles every 18 months**, e.g. measured by the number of transistors
- **But** this can not be continued arbitrarily
- Physical limits in the manufacturing of semiconductors
- By now there are transistors with a size of approx. 14 nm
- Compared to the wavelength of visible light 400 - 700 nm

# Moor's law I

- The complexity of integrated circuits **doubles every 18 months**, e.g. measured by the number of transistors
- **But** this can not be continued arbitrarily
- Physical limits in the manufacturing of semiconductors
- By now there are transistors with a size of approx. 14 nm
- Compared to the wavelength of visible light 400 - 700 nm
- "Computers are not getting faster but wider."
- Solution may be **cluster computers** with fast interconnects

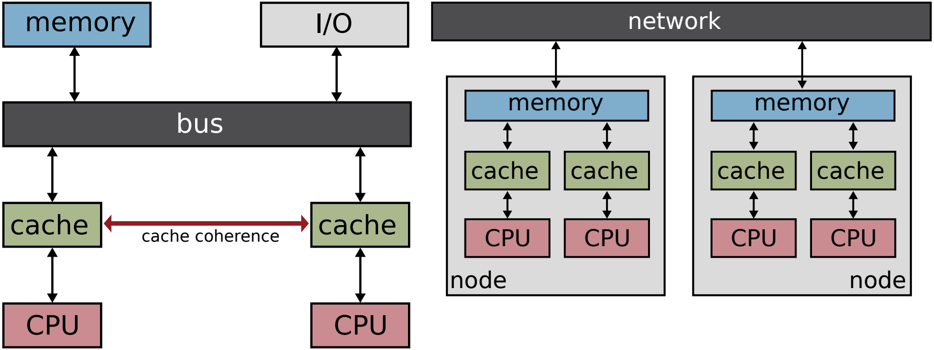



Source: `http://de.wikipedia.org/wiki/Mooresches_Gesetz`




- **Shared memory system**
- Laptops, smartphones . . .

# Shared and distributed memory systems




- **Shared memory system**
- Laptops, smartphones . . .

- **Distributed memory system**
- Large webservers, databases, supercomputers

# Top 500 list of super computers

**Top 10 positions of the 46th TOP500 in November 2015**

| Rank | Rmax Rpeak (PFLOPS) | Name | Computer design Processor type, interconnect | Vendor | Site Country, year | Operating system |
|---|---|---|---|---|---|---|
| **1** | 33.863 54.902 | *Tianhe-2* | **NUDT** Xeon E5–2692 + Xeon Phi 31S1P, TH Express-2 | NUDT | National Supercomputing Center in Guangzhou China, 2013 | Linux (Kylin) |
| **2** | 17.590 27.113 | *Titan* | **Cray XK7** Opteron 6274 + Tesla K20X, Cray Gemini Interconnect | Cray Inc. | Oak Ridge National Laboratory United States, 2012 | Linux (CLE, SLES based) |
| **3** | 17.173 20.133 | *Sequoia* | **Blue Gene/Q** PowerPC A2, Custom | IBM | Lawrence Livermore National Laboratory United States, 2013 | Linux (RHEL and CNK) |
| **4** | 10.510 11.280 | *K computer* | **RIKEN** SPARC64 VIIIfx, Tofu | Fujitsu | RIKEN Japan, 2011 | Linux |
| **5** | 8.586 10.066 | *Mira* | **Blue Gene/Q** PowerPC A2, Custom | IBM | Argonne National Laboratory United States, 2013 | Linux (RHEL and CNK) |
| **6** | 8.101 11.079 | *Trinity* | **Cray XC40** Xeon E5-2698v3, Aries | Cray Inc. | DOE/NNSA/LANL/SNL United States, 2015 | Linux (CLE) |
| **7** | 6.271 7.779 | *Piz Daint* | **Cray XC30** Xeon E5–2670 + Tesla K20X, Aries | Cray Inc. | Swiss National Supercomputing Centre Switzerland, 2013 | Linux (CLE) |
| **8** | 5.640 7.404 | *Hazel Hen* | **Cray XC40** Xeon E5-2680v3, Aries | Cray Inc. | High Performance Computing Center, Stuttgart Germany, 2015 | Linux (CLE) |
| **9** | 5.537 7.235 | *Shaheen II* | **Cray XC40** Xeon E5–2698v3, Aries | Cray Inc. | King Abdullah University of Science and Technology Saudi Arabia, 2015 | Linux (CLE) |
| **10** | 5.168 8.520 | *Stampede* | **PowerEdge C8220** Xeon E5–2680 + Xeon Phi, Infiniband | Dell | Texas Advanced Computing Center United States, 2013 | Linux (CentOS)[13] |

Source: `https://en.wikipedia.org/wiki/TOP500`

Cray XC40 at HLRS Stuttgart

| | |
|---|---|
| Peak performance | 7420 TFlops |
| Weight | 61.5 T |
| Compute nodes | 7712 each with 2x 12 cores (185,088 cores) |
| Memory per node | 128 GB |
| Power consumption | $\approx$ 3.2 MW |

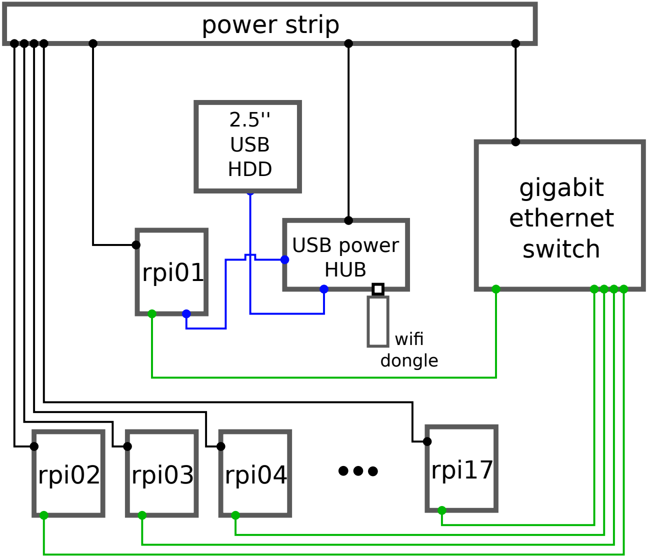
**Source**: `https://www.hlrs.de/en/systems/cray-xc40-hazel-hen/`

# Super computer in a nutshell

Typical components of supercomputers we have to imitate:

1. The **frontend**
   - A dedicated login node managing user interaction
   - Accessible from the outside world (e.g. via `ssh`)
   - Manage data, compile code, place your program run in the execution queue
2. The **backend**
   - Nodes dedicated for computing only, not directly accessible
3. **Input/Output devices**
   - Parallel IO is problematic, not touched in our small setup

# A blueprint of the hardware



# Material needed

| | amount | price (euro) |
|---|---|---|
| Raspberry Pi 2 B | 17 | 38.00 |
| Micro SD 16GB | 17 | 5.00 |
| UBS power charger | 17 | 8.00 |
| Network cable | 17 | 0.50 |
| RPi cases | 17 | 6.00 |
| Ethernet switch 24 port | 1 | 150.00 |
| USB power HUB | 1 | 25.00 |
| 2.5" USB HDD 1TB | 1 | 55.00 |
| Wifi dongle | 1 | 10.00 |
| Wood, screws, cable ties . . . | | 35.00 |
| | | 1252.50 |

# Installation instructions

- Basically, we made 2 Raspbian Wheezy installations
  - The login node
  - One compute node that is cloned 16 times
- Distribution does not matter
- **However**, be sure to have hard float support
- On both installation we create the user `pi`
- We **do not want** to copy our program 16x whenever it is changed
- All compute nodes have to **share** the same `/home/pi` folder

On the server we need the following packages:

1. `usbmount`
   - Automatically mount the extern USB HDD after boot
2. openssh-server
   - Access the server without keyboard or monitor
   - Passwordless authentication is required → later
3. `nfs-kernel-server`
   - Share the home folder on the USB HDD to the compute nodes
4. `ntp`
   - Server must receive time from the internet and provide it to the compute nodes
   - For some scenarios it is important that all compute nodes precisely share the same time
5. `gcc`, `g++`, `gfortran`, `openmpi` ...

# Network share

1. On the compute node we install `nfs-common`, `openssh-server`, `openmpi`
2. On server side modify `/etc/exports` to

```
/media/extern_usb/ 192.168.0.1/24(rw,fsid=0,insecure,no_subtree_check,async)
/media/extern_usb/pi_home 192.168.0.1/24(rw,nohide,insecure,no_subtree_check,async)
```

3. On client side add the following to `/etc/fstab`

```
192.168.0.1:/pi_home /home/pi nfs
rw,nouser,atime,_netdev,dev,hard,intr,rsize=8192,wsize=8192 0 2

192.168.0.1:/pi_opt /opt nfs
ro,nouser,atime,_netdev,dev,hard,intr,rsize=8192,wsize=8192 0 2
```

4. In `/etc/ntp.conf` add the login node as ntp server

# Getting to know each other

In the file /etc/hosts we add the following lines to get rid of IP addresses:

```
192.168.0.1     rpi01
192.168.0.2     rpi02
192.168.0.3     rpi03
192.168.0.4     rpi04
192.168.0.5     rpi05
192.168.0.6     rpi06
192.168.0.7     rpi07
192.168.0.8     rpi08
192.168.0.9     rpi09
192.168.0.10    rpi10
192.168.0.11    rpi11
192.168.0.12    rpi12
192.168.0.13    rpi13
192.168.0.14    rpi14
192.168.0.15    rpi15
192.168.0.16    rpi16
192.168.0.17    rpi17
```

# Cloning the compute nodes

Time to **clone** the compute nodes!

1. Copy the content of `/home/pi` to the USB HDD and then **delete it locally**
2. **Clone the SD card** for each compute node
   - Use `dd` for that
   - After each cloning mount the second partition of the SD card and adjust the file `/etc/hostname` to, e.g. `rpi09` for the 9th compute node
   - Since we use static IP addresses we have to adjust the file `/etc/network/interfaces` on each compute node to the correct IP
3. **Finally**, put the cluster computer together and start the engine

## Changing IP address

- The login node uses wifi to connect to campus network
- On each boot it gets a different IP address

## Changing IP address

- The login node uses wifi to connect to campus network
- On each boot it gets a different IP address

## Dynamic DNS

- We use dynamic DNS to map the current IP to a global domain name
- Free services like `afraid.org`
- We use `hpc-workshop.mooo.com`

## Changing IP address

- The login node uses wifi to connect to campus network
- On each boot it gets a different IP address

## Dynamic DNS

- We use dynamic DNS to map the current IP to a global domain name
- Free services like `afraid.org`
- We use `hpc-workshop.mooo.com`

## Two cron jobs solve access problem

```
* * * * * wget --no-check-certificate -O - update-url >> /tmp/dns.log 2>&1 &
* * * * * touch /media/extern_usb/pi_home/.stayingalive &> /dev/null
```

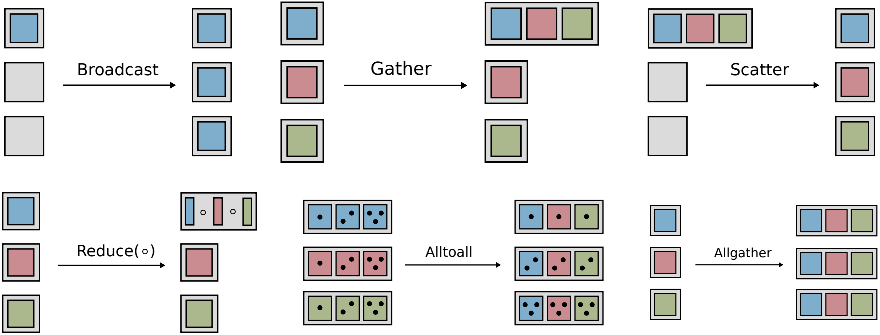
Passwordless SSH connections are mandatory:

- By now we are asked for a password to log into the cluster
- If we start a parallel program, we would be asked for a password **for every compute node**

### SSH pubic key

- `ssh-keygen -t rsa -b 4096` on login node
- Enter **empty password** (think twice when and where you do this)
- Use `ssh-copy-id` to bring the public part of the key to `rpi02` (and thereby to all compute nodes)

- Data exchanges between processors are conducted via **Message Passing Interface** (MPI)
- We use the **OpenMPI** C library as MPI implementation
- The interface describes a collection of basic exchange routines
- Besides point-to-point send and receive we have:



A few examples:

- Parallel calculation of fractals
- Load balancing by graph partitioning
- Option pricing with Monte Carlo methods
- Large scale linear systems

Informal definitions:

### Strong scalability

- **Fixed problem size**
- Number of procs is increased
- Good scalability: #proc is doubled and runtime is halved

# What does scalability mean?

Informal definitions:

## Strong scalability

- **Fixed problem size**
- Number of procs is increased
- Good scalability: #proc is doubled and runtime is halved

## Weak scalability

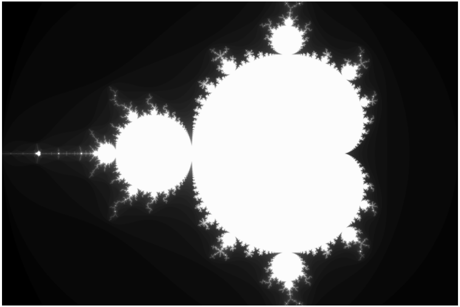
- Most interesting value for supercomputers
- **Problem size** and **number of procs** is increased
- Desirable result: #proc and problem size are doubled, runtime is constant

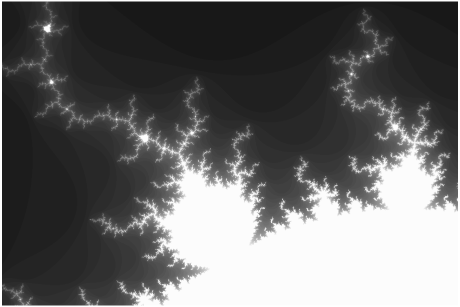
# Perfect scalability: Mandelbrot set

Examine boundedness for $c \in \mathbb{C}$ of

$$z_{n+1} = z_n^2 + c\,,\ z_0 = 0.$$

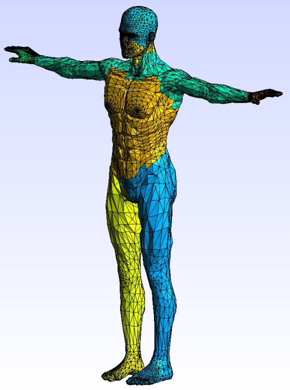
$$\text{Is } z_n \text{ for } n \to \infty \text{ bounded } \begin{cases} \text{yes? } c \text{ becomes white} \\ \text{no? } c \text{ becomes black} \end{cases}$$

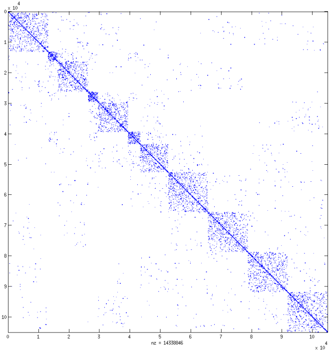




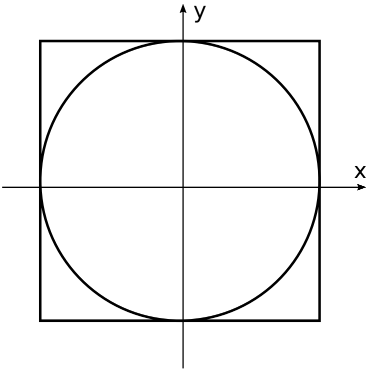
# Load balancing

- Partition nodes such that each processor has the **same load**
- Cut edges result in **network communication** → **minimize this**
- Example: Simulate elastic phenomena in human body
- Geometric model and corresponding matrix system

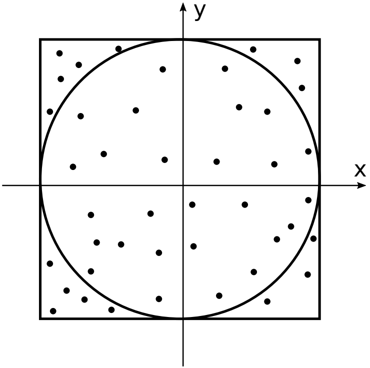




- Unit sphere in $\mathbb{R}^d$ is given by
  $S^d = \{z \in \mathbb{R}^d : \|z\|_2 \leq 1\}$

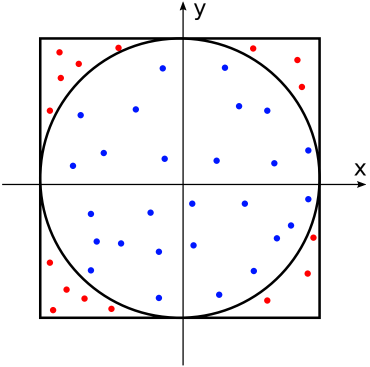


- Unit sphere in $\mathbb{R}^d$ is given by $S^d = \{z \in \mathbb{R}^d : \|z\|_2 \leq 1\}$
- Generate uniformly distributed points in $z \in [-1, 1]^d$



- Unit sphere in $\mathbb{R}^d$ is given by $S^d = \{z \in \mathbb{R}^d : \|z\|_2 \leq 1\}$
- Generate uniformly distributed points in $z \in [-1, 1]^d$
- Volume of $S^d$ is approximated by

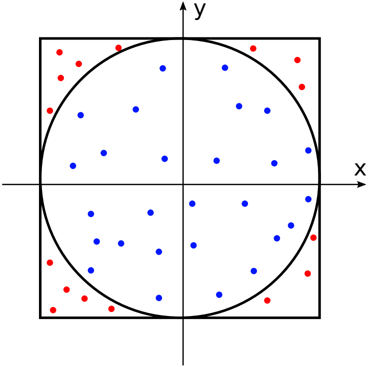
$$Vol(S^d) \approx 2^d \cdot \frac{\#\text{ dots outside}}{\#\text{ dots inside}}$$



# Volume of the d-dimensional sphere

- Unit sphere in $\mathbb{R}^d$ is given by $S^d = \{z \in \mathbb{R}^d : \|z\|_2 \leq 1\}$
- Generate uniformly distributed points in $z \in [-1, 1]^d$
- Volume of $S^d$ is approximated by

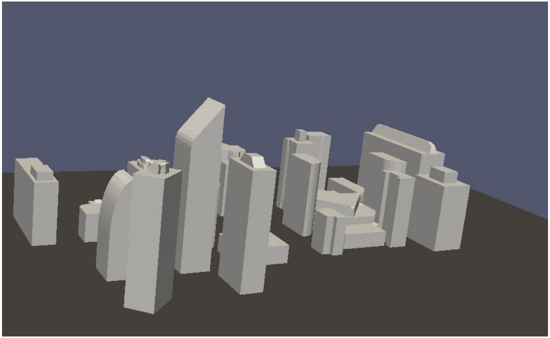
$$Vol(S^d) \approx 2^d \cdot \frac{\#\text{ dots outside}}{\#\text{ dots inside}}$$

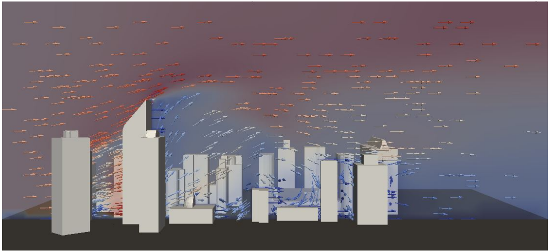



## Almost perfect scalability

- Each compute node works independently on $N$ points
- Compute parallel sum of number of red dots

- Simulation of **fluid flows** through a city
- Leads to the solution of **large linear systems**
- **Moderate scalability** on the Pi cluster
- **Slow network** is the bottleneck





# Conclusion

What this cluster is not:

- Not suitable for **real world problems** due to the **slow network**

Things the students learned:

- Many facets of **Linux** administration and networks
- How to **implement mathematical methods** on a computer using C++ and MPI
- Pitfalls in parallelizing numerical algorithms