CME 213

Spring 2017

Eric Darve

Stanford University

Syllabus

PEOPLE

Instructors:

- **Eric Darve, ME, ICME**
- **Colfax and NVIDIA engineers**

Teaching assistants:

- **Qing Wang, wangqing@stanford.edu**
- **Ruoxi Wang, ruoxi@stanford.edu**

Web sites

- **piazza:**
  - Forum discussion, Q&A
  - You need to register

- **canvas.stanford.edu**
  - Homework upload
  - Lecture notes, reading material
  - Computer code, etc

GRADING, HOMEWORK, PROJECT

- 1 pre-requisite homework + 4 homework assignments: 65% of grade
- One final project: 35% of grade
- The first homework is due Wednesday April 12.

- Honor code:

  > "that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading"

- You cannot copy someone's computer code.
- The work you submit must be your own.

COMPUTERS

- For the first few homeworks, you can use FarmShare or your own computer. You need a standard C/C++ compiler.

https://web.stanford.edu/group/farmshare/cgi-bin/wiki/index.php/Main_Page

- You will also have access to the computer cluster Certainty.
- Certainty has GPUs which are needed for Homework 3 and after.

How to use Certainty

- You need to use VPN for Certainty.

`https://uit.stanford.edu/service/vpn`

- To login:

`ssh darve@certainty-login.stanford.edu`

- Information on cluster:

https://hpcc-intranet.stanford.edu/resources/

- Check "How do I…?" for useful information.

TIPS AND TRICKS

- How to log without a password?
- That's legit!

STEP 1

- Say you want to log in from Host A / User a to Host B / User b.
- On A, generate a pair of authentication keys. Do not enter a passphrase:

```
a@A:~> ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/a/.ssh/id_rsa):
Created directory '/home/a/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/a/.ssh/id_rsa.
Your public key has been saved in /home/a/.ssh/id_rsa.pub.
The key fingerprint is:
3e:4f:05:79:3a:9f:96:7c:3b:ad:e9:58:37:bc:37:e4 a@A
```

Step 2

- The `.pub` file is safe to share because this is your public key.
- On B, create the directory `~/.ssh`
- Append your public key created on A to `authorized_keys` on B.

```
a@A:~> cat .ssh/id_rsa.pub | ssh b@B 'cat >> .ssh/authorized_keys'
```

RUNNING A CODE ON CERTAINTY

See

https://hpcc-intranet.stanford.edu/resources/using-the-hpc-clusters-101/

Step 1: load modules with compiler and libraries

$ module avail

$ module list

$ module load intel/16

```
[darve@certainty-c ~]$ module avail

------------------------------------------------------------------ /share/apps/modules/modulefiles -------------------------------------------
ansys/15.0                                  gnu/4.9.2                                   openmpi/1.8.6-intel-15
ansys/16.1                                  gnu/5.2                                     paraview/4.1.0
ansys/18.0                                  hdf5/1.8.13-mvapich2-2.0rc1-intel-14        parmetis/4.0.3-mvapich2-2.0rc1-intel-14
boost/1.55.0-mvapich2-2.0rc1-intel-14       hdf5/1.8.17-mvapich2-2.2b-intel-16          parmetis/4.0.3-mvapich2-2.1rc1-intel-15
boost/1.63.0-mvapich2-2.2b-intel-16         intel/14                                    parmetis/4.0.3-mvapich2-2.2b-intel-16
clang/3.5.2                                 intel/15                                    petsc/3.0.0-p12-mvapich2-2.0rc1-intel-14
cmake/2.8.12.2                              intel/16                                    petsc/3.0.0-p12-mvapich2-2.0rc1-intel-14-DEBUG
comsol/44                                   maple/2016                                  petsc/3.0.0-p12-mvapich2-2.1rc1-intel-15
comsol/51                                   matlab/R2014a                               petsc/3.4.4-mvapich2-2.0rc1-intel-14
comsol/52                                   matlab/R2015b                               petsc/3.4.4-mvapich2-2.0rc1-intel-14-DEBUG
convergecfd/2.2.0                           mvapich2/2.0rc1-intel-14                    petsc/3.7.2-mvapich2-2.2b-gnu-4.9.2
cuda/7.0.28                                 mvapich2/2.1a-gcc-4.4.7                     petsc/3.7.2-mvapich2-2.2b-gnu-5.2
cuda/7.5.18                                 mvapich2/2.1-intel-15                       pointwise/18.0R1
dakota/6.2.0                                mvapich2/2.1rc1-intel-15                    python/2.7.8
ensight/10.1.6b                             mvapich2/2.2b-gnu-4.4.7-11                  scons/2.5.0
freefem++/3.30                              mvapich2/2.2b-gnu-4.7.2                     simvascular/2016-04-06
freefem++/3.38-1                            mvapich2/2.2b-gnu-4.9.2                     tecplot/14.0.2
freefem++/3.46-gnu-4.9.2                    mvapich2/2.2b-gnu-5.2                       tecplot/14.1.0.51525
freefem++/3.47-gnu-4.9.2                    mvapich2/2.2b-intel-16                      tecplot/2016-R2-EX
freefem++/3.47-mvapich2-2.2b-gnu-5.2        null                                        totalview/2016.01.0
```

RUNNING A CODE ON CERTAINTY

Step 2:

- Compile your code
- Submit a job using **qsub** to the GPU queue:

```
$ qsub -q gpu submit.sh
$ qstat –q gpu
$ qstat –u darve
```

EXAMPLE OF SCRIPT: SUBMIT.SH

```
#!/bin/bash

#PBS -N cme213
#PBS -e cme213.err
#PBS -o cme213.out
#PBS -l nodes=1:ppn=24
#PBS -V

PBS_O_WORKDIR='/home/darve/CME213/Lecture_03/code'
export PBS_O_WORKDIR

### ---------------------------------------
### BEGINNING OF EXECUTION
### ---------------------------------------

echo The master node of this job is `hostname`
echo The working directory is `echo $PBS_O_WORKDIR`
echo This job runs on the following nodes:
echo `cat $PBS_NODEFILE`
echo
echo Output from code
echo ----------------

### end of information preamble

cd $PBS_O_WORKDIR
./matrix_prod -n 512 -p 24
```

The final project

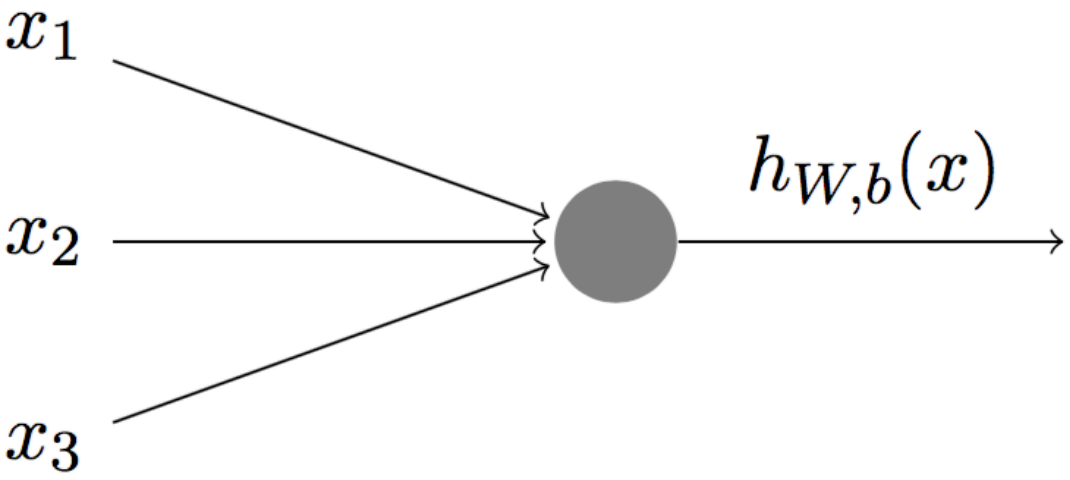
- You will be given a final project to work on towards the end of the quarter.
- Everyone will work on the same project.
- The final project will be on neural network for machine learning.
- The project will involve multi-GPU programming with CUDA and MPI.

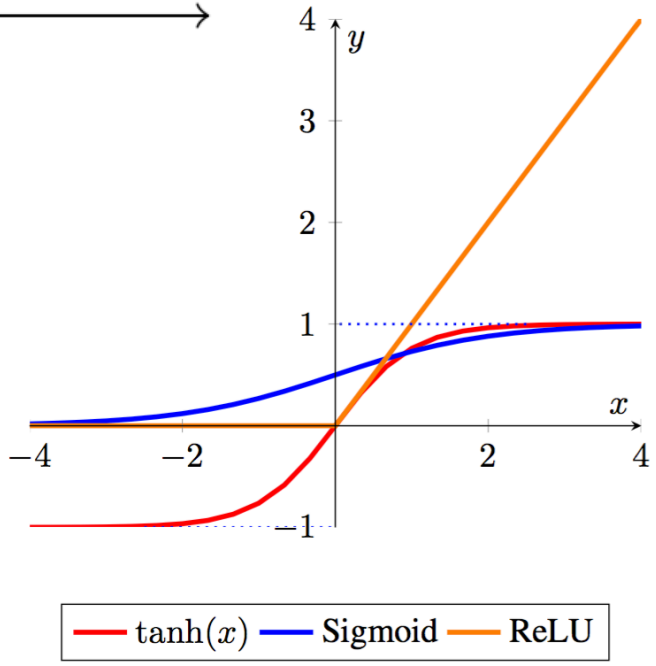
Neural networks

- **Simple but powerful**
- **"Mimic" neurons in the brain: a network of nodes that receive inputs and generate outputs.**

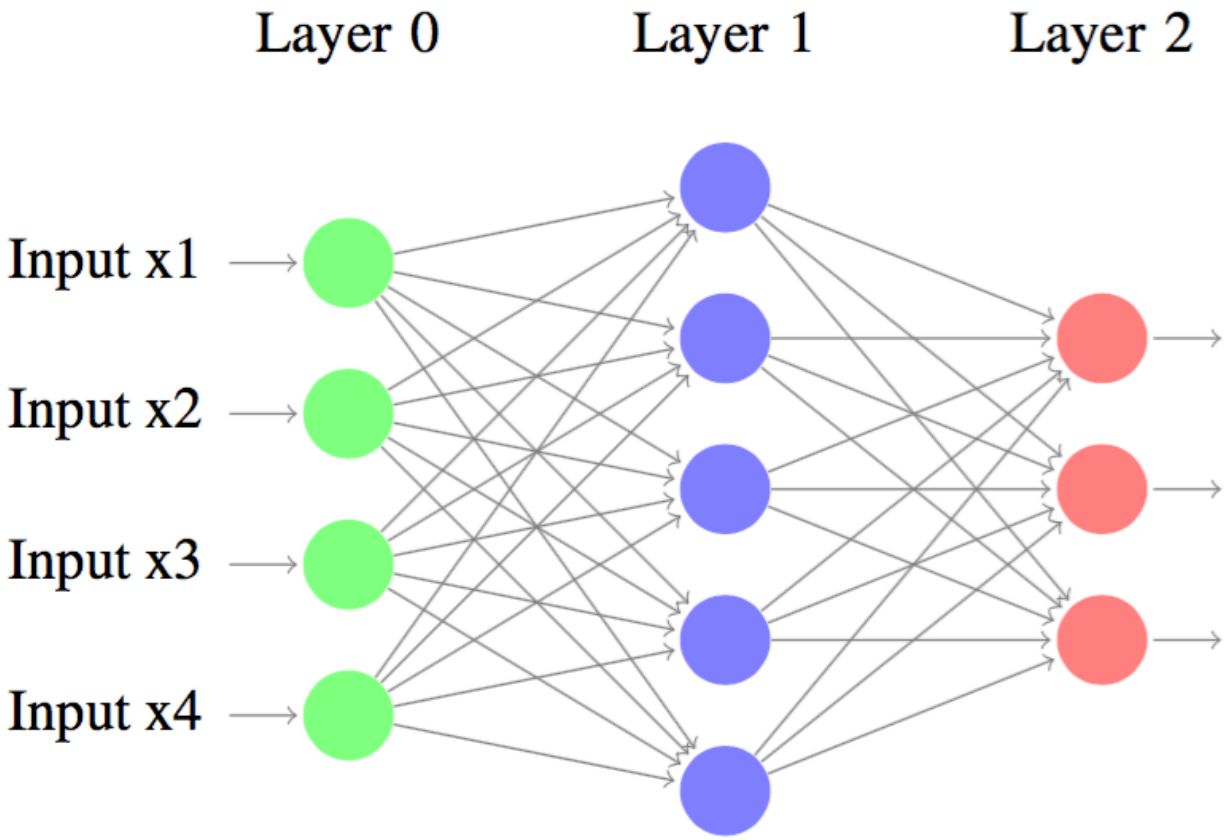
WHAT'S A NEURON EXACTLY?

- **It's a highly simplified model of a biological neuron.**





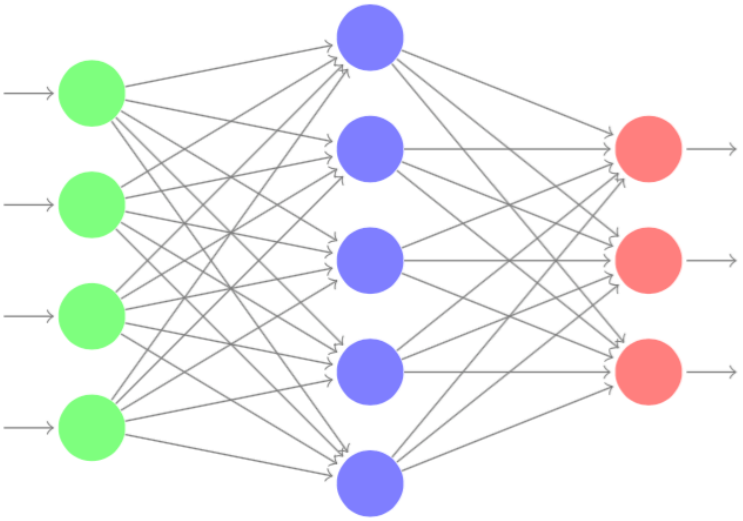
Assemble many neurons to create a neural network.



Input to neural network: handwritten digit

OUTPUT: PROBABILITY OF DIGIT

- The network gives a probability vector of size 10
- Entry i is the probability that the image is showing the digit i.



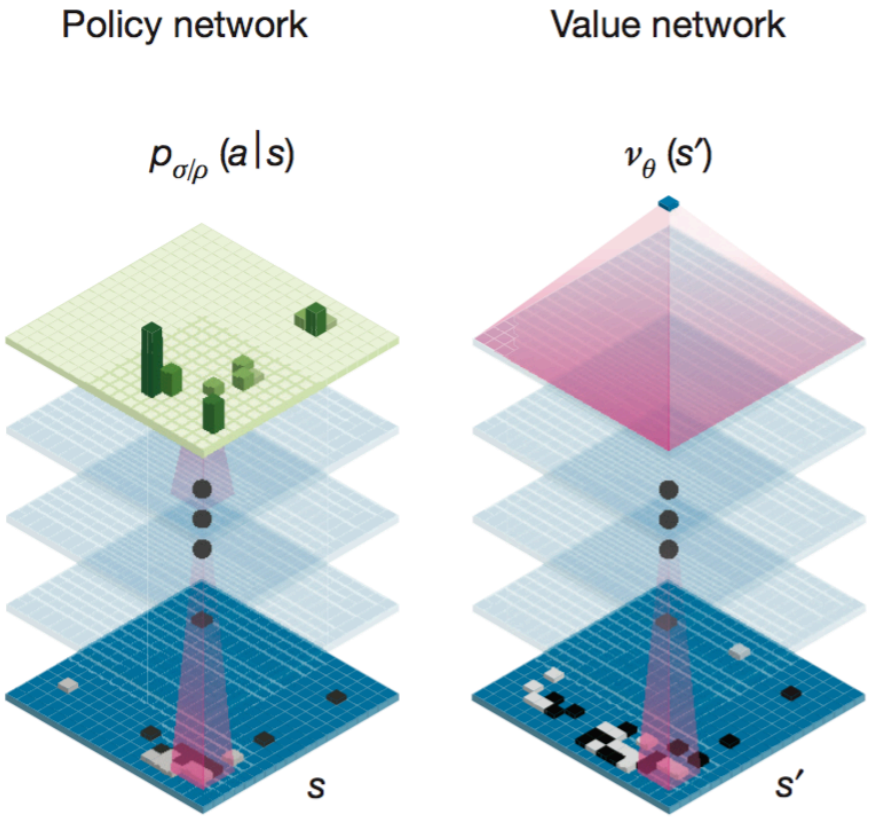
P(0)
P(1)
P(2)
P(3)
P(4)
P(5)
P(6)
P(7)
P(8)
P(9)

AlphaGo

- AlphaGo (a computer program written by Google DeepMind) took on and defeated legendary Go player, Lee Sedol (9-dan professional with 18 world titles).
- The final score was 4 to 1.

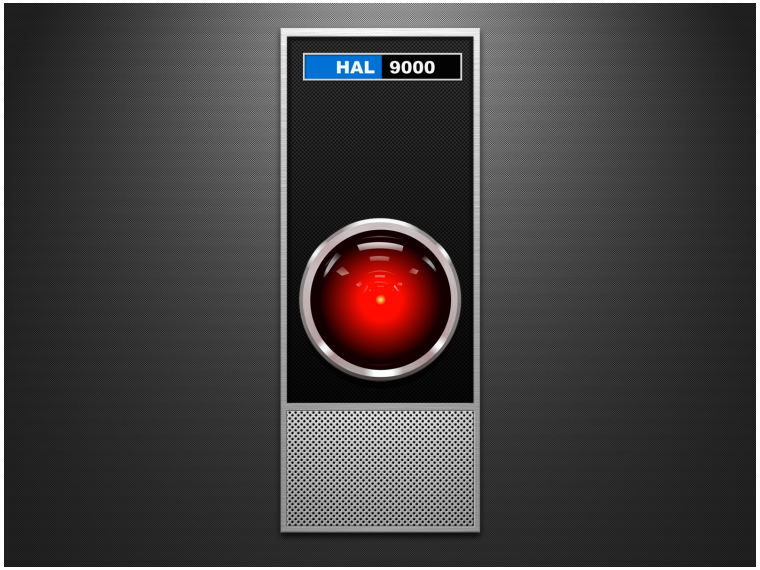
AlphaGo

- We will talk more about this later in the quarter.
- DeepMind used deep learning to create a program to play Go.
- It uses parallel computing and GPUs to use neural networks to play Go.
- Hardware: 1,202 CPUs and 176 GPUs.



"Although we have programmed this machine to play, we have no idea what moves it will come up with. Its moves are an emergent phenomenon from the training. We just create the data sets and the training algorithms. But the moves it then comes up with are out of our hands—and much better than we, as Go players, could come up with."

DeepMind research scientist Thore Graepel



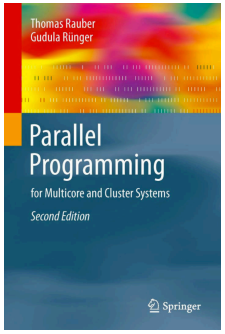
Books

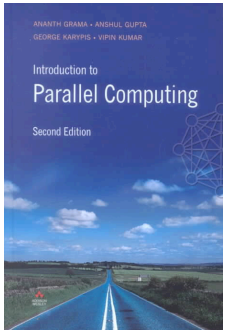
Good news: all books are available electronically from the Stanford Library. Just go to:

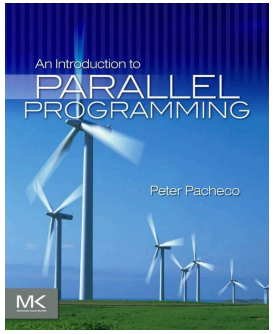
http://searchworks.stanford.edu/

OpenMP, MPI, Parallel programming

- *Parallel Programming for Multicore and Cluster Systems*, Rauber and Rünger. Applications focus mostly on linear algebra.
- *Introduction to Parallel Computing*, Grama, Gupta, Karypis, Kumar. Wide range of applications from sort to FFT, linear algebra and tree search.
- *An introduction to parallel programming*, Pacheco. More examples and less theoretical. Applications include n-body codes and tree search.

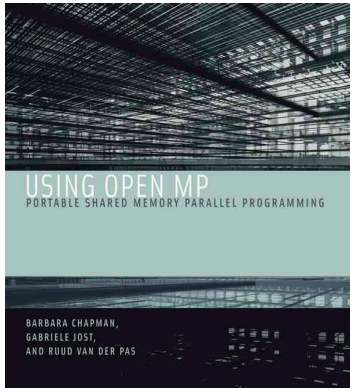


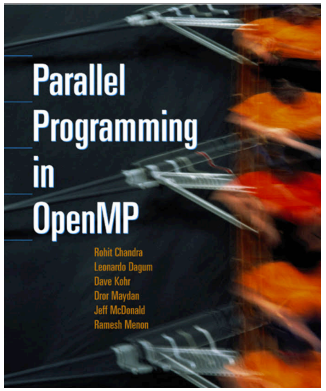


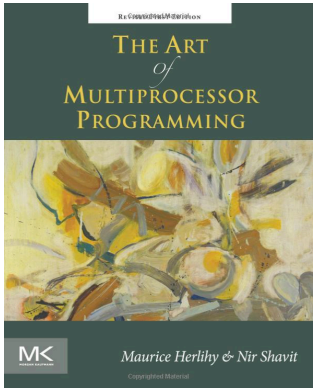


OpenMP and Multicore Books

- *Using OpenMP: portable shared memory parallel programming*, Chapman, Jost, van der Pas. Advanced coverage of OpenMP.
- *Parallel Programming in OpenMP*, Chandra, Menon, Dagum, Kohr, Maydan, McDonald; a bit outdated.
- *The art of multiprocessor programming*, Herlihy, Shavit. Specializes on advanced multicore programming.

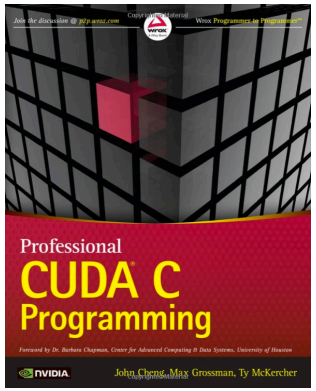


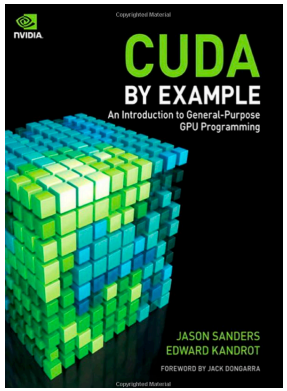


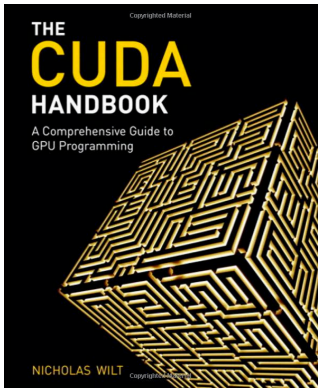


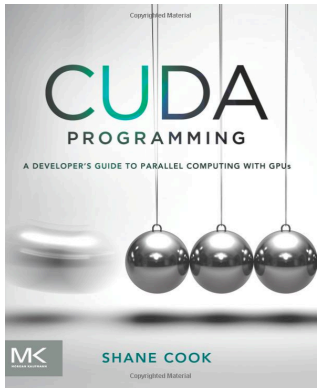
CUDA BOOKS

- *Professional CUDA C Programming*, Cheng, Grossman, McKercher; recent book, recommended for this class
- *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Sanders, Kandrot
- *CUDA Handbook: A Comprehensive Guide to GPU Programming*, Wilt
- *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*, Cook









CUDA BOOKS: PREFERRED REFERENCES

- CUDA online documentation:

http://docs.nvidia.com/cuda/index.html

https://developer.nvidia.com/cuda-education-training

- Uploaded reading material:

CUDA_C_Best_Practices_Guide.pdf

CUDA_C_Programming_Guide.pdf

WHAT THIS CLASS IS ABOUT

- **We will focus on how to program:**
  - **Multicore processors, e.g., desktop processors: Pthreads, OpenMP.**
  - **Introduction to Xeon Phi and Knights Landing**
  - **NVIDIA graphics processors using CUDA.**
  - **Computer clusters using MPI.**
- **We will cover some numerical algorithms for illustration: sort, linear algebra, basic parallel primitives.**

What this class is not about

- **Parallel computer architecture**
- **Parallel design patterns and programming models**
- **Parallel numerical algorithms. See *CME 342: Parallel Methods in Numerical Analysis***

What this class requires

- Some basic knowledge of UNIX (ssh, makefile, etc)
- Good knowledge of C and C++ (including pointers, templates)
- Proficiency in scientific programming, including debugging and testing

Why Parallel Computing?

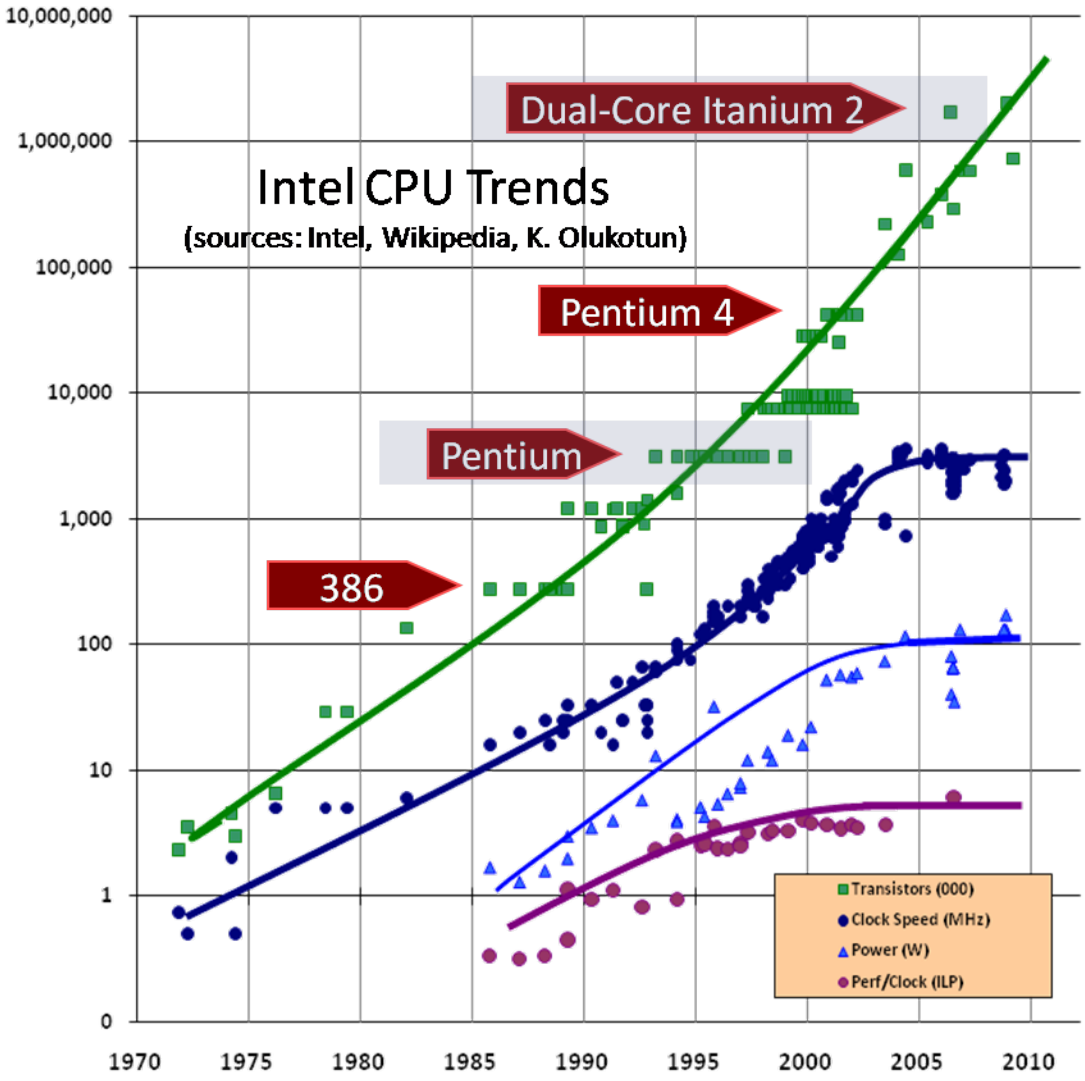
WHY PARALLEL COMPUTING?

- Parallel computing has existed for a long time but until recently it was a specialized area that concerned only a small fraction of engineers.
- Nowadays, parallel computing is a dominant player in scientific and large scale computing.
- What happened?

WHY PARALLEL COMPUTING?

- Gordon Moore 1965: the number of transistors on a chip shall double every 18-24 months.
- This has been valid for more than 40 years.
- This increase in the number of transistors has been accompanied by an increase in clock speed.

Intel microprocessor trends



Performance increase

- Increase in transistor density is limited by:
  - Leakage current increases
  - Power consumption increases
  - Heat generated increases
- In addition, the memory access time has not been reduced at a rate comparable with processing speed (processor clock period).
- New ways need to be found.
- The most promising approach is to have multiple cores on a single processor.

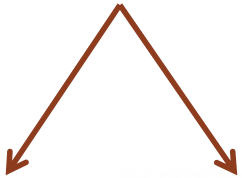
SEQUENTIAL VS PARALLEL

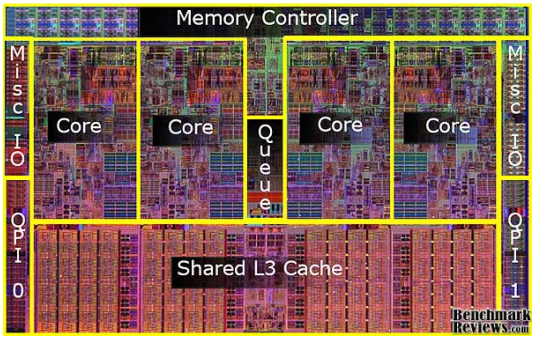
The Sequential Giant
Build bigger, meaner, faster processor

The Parallel Ants
Lots of smaller, slower cores

PARALLEL COMPUTING EVERYWHERE: MOBILE DEVICES

Laptops and desktops

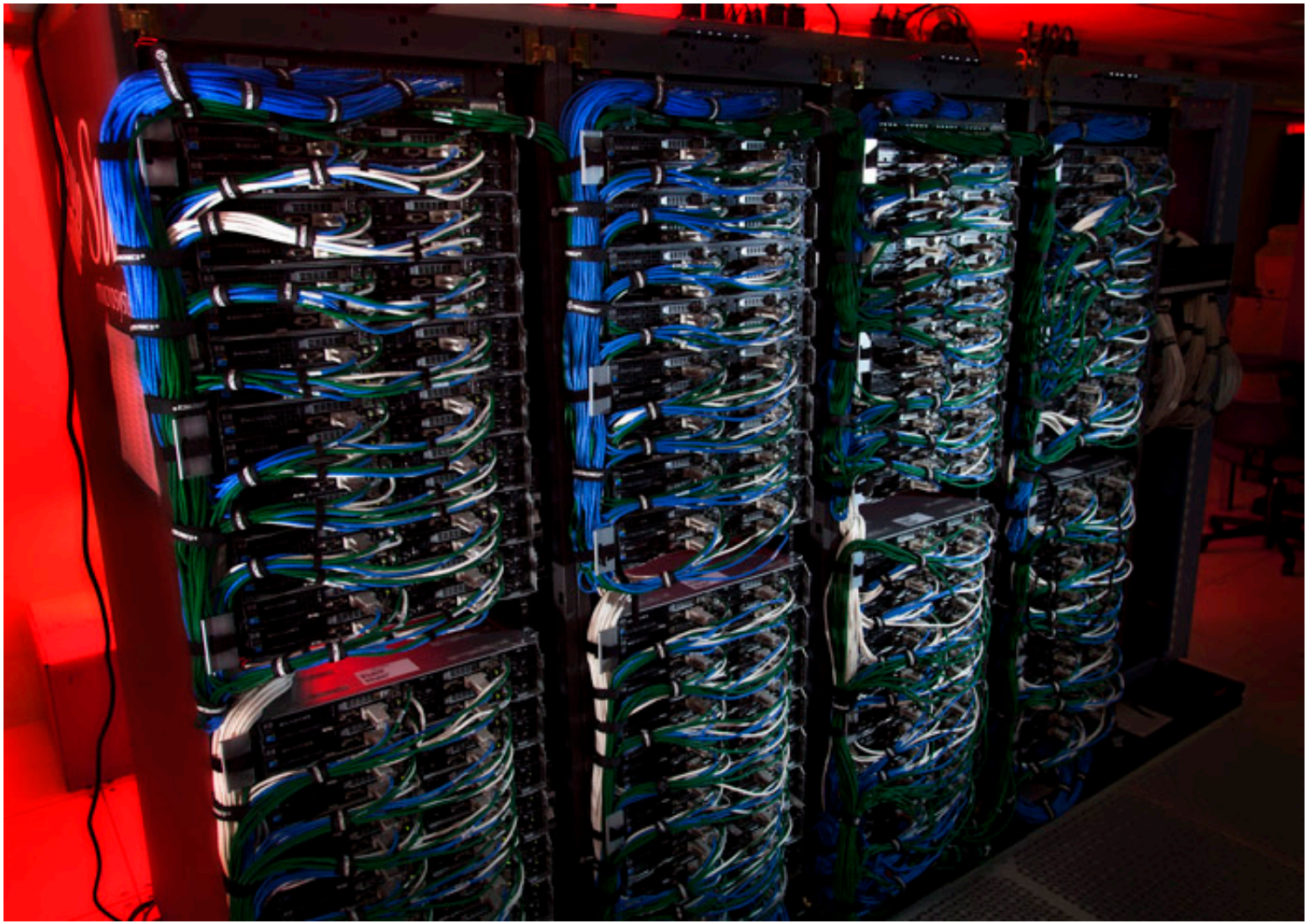




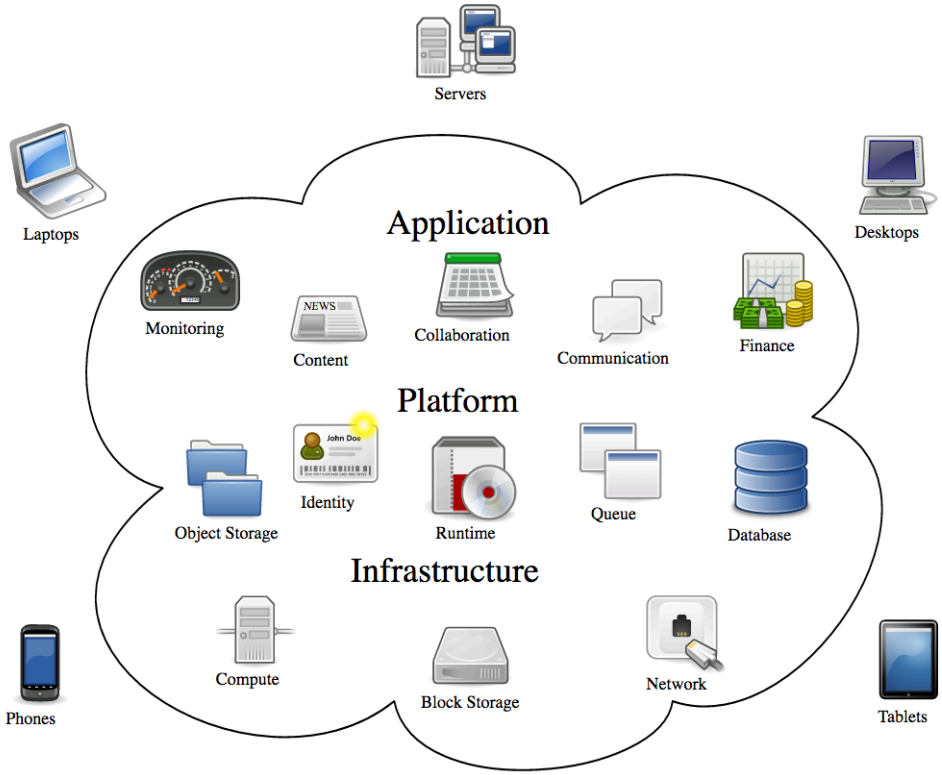
multicore

GPU

Clusters



CLOUD COMPUTING



Cloud computing

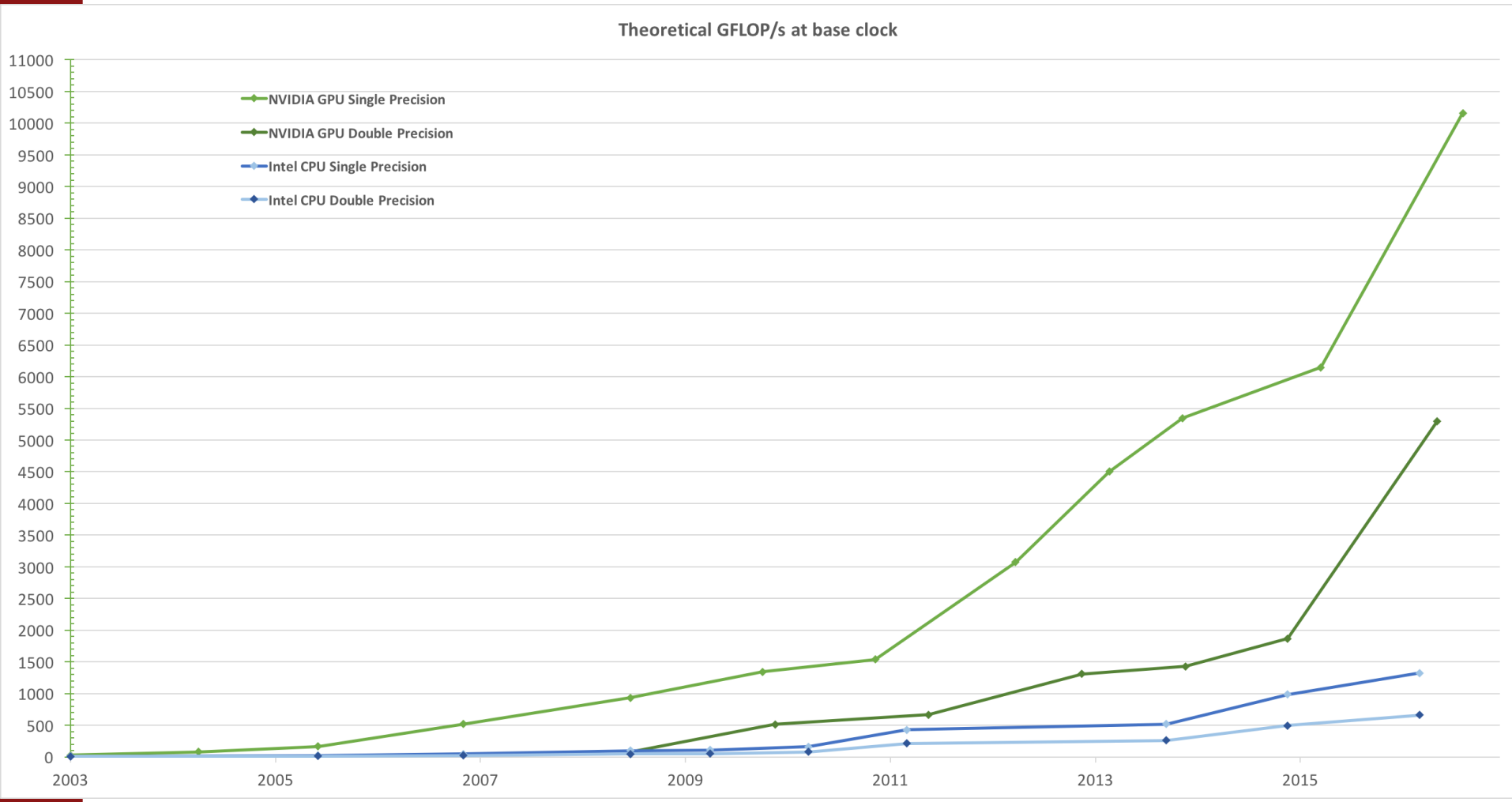
Cloud computing is a type of Internet-based computing that provides shared computer processing resources and data to computers and other devices on demand.

SUPERCOMPUTERS

- Titan
- Cray Supercomputer, Oak Ridge National Lab

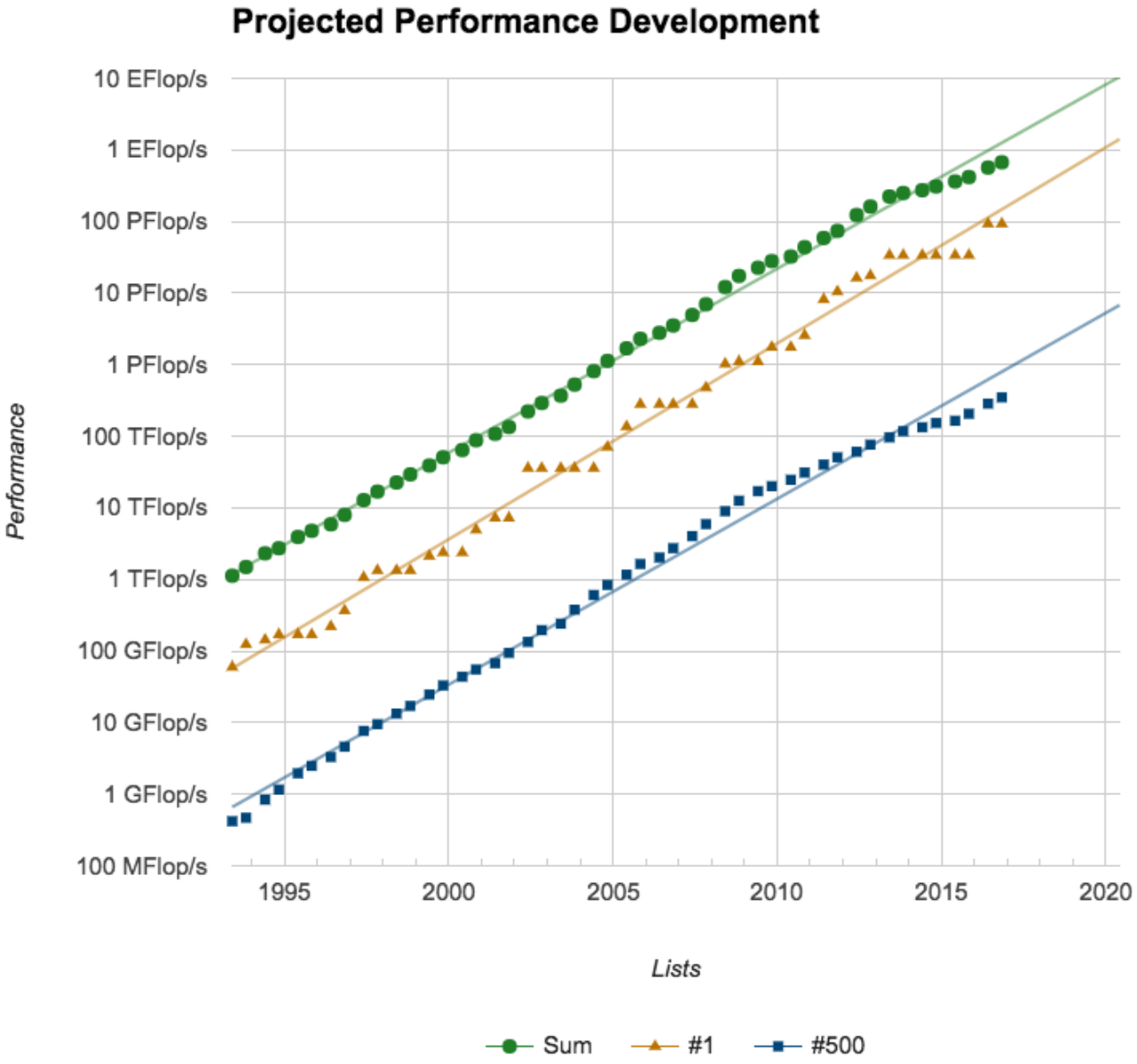
Active	Became operational October 29, 2012
Sponsors	US DOE and NOAA (<10%)
Operators	Cray Inc.
Location	Oak Ridge National Laboratory
Architecture	18,688 AMD Opteron 6274 16-core CPUs 18,688 Nvidia Tesla K20X GPUs
Power	8.2 MW
Operating system	Cray Linux Environment
Space	404 m^2 (4352 ft^2)
Memory	693.5 TiB (584 TiB CPU and 109.5 TiB GPU)
Storage	40 PB, 1.4 TB/s IO Lustre filesystem
Speed	17.59 petaFLOPS (LINPACK) 27 petaFLOPS theoretical peak
Cost	$97 million
Ranking	TOP500: #3, June 2016[1]

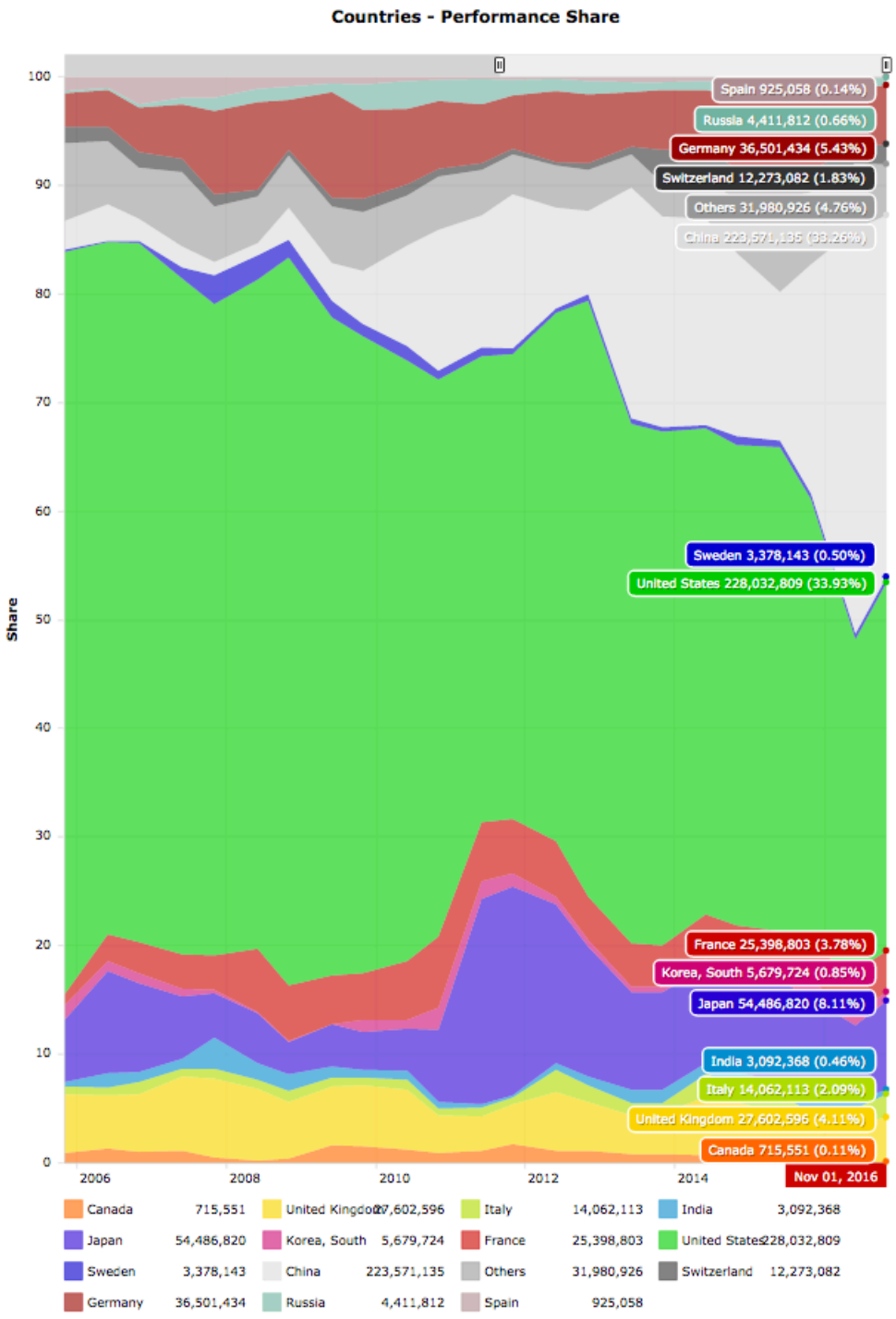
PERFORMANCE THROUGH PARALLEL PROCESSING



Statistics about the top 500 Supercomputers

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
3	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
4	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
5	DOE/SC/LBNL/NERSC United States	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect Cray Inc.	622,336	14,014.7	27,880.7	3,939
6	Joint Center for Advanced High Performance Computing Japan	**Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path Fujitsu	556,104	13,554.6	24,913.5	2,719


Projected Performance Development
10 EFlop/s
1 EFlop/s
100 PFlop/s
10 PFlop/s
1 PFlop/s
100 TFlop/s
10 TFlop/s
1 TFlop/s
100 GFlop/s
10 GFlop/s
1 GFlop/s
100 MFlop/s
Performance
1995
2000
2005
2010
2015
2020
Lists
Sum
#1
#500



Stanford University