Lecture 1: An Introduction Parallel Computing CSCE 569, Spring 2018

Department of Computer Science and Engineering
Yonghong Yan
yanyh@cse.sc.edu
http://cse.sc.edu/~yanyh

Course Information

- **Meeting Time:** 9:40AM – 10:55AM Monday Wednesday
- **Class Room:** 2A15, Swearingen Engineer Center, 301 Main St, Columbia, SC 29208
- **Grade: 60**% for four homeworks + 40% for two exams

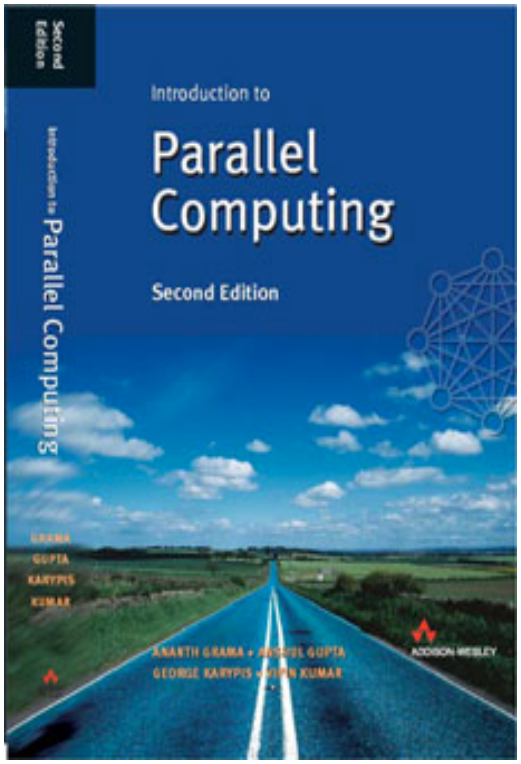
- **Instructor:** Yonghong Yan
  - http://cse.sc.edu/~yanyh, yanyh@cse.sc.edu
  - **Office:** Room 2211, Storey Innovation Center (Horizon II), 550 Assembly St, Columbia, SC 29201
  - **Tel:** 803-777-7361
  - ***Office Hours: 11:00AM – 12:30AM (after class) or by appointment***

- **Public Course website**: http://passlab.github.io/CSCE569
- **Homework submission:** https://dropbox.cse.sc.edu
- **Syllabus or website** for more details

Objectives

- Learn fundamentals of concurrent and parallel computing
  - Describe benefits and applications of parallel computing.
  - Explain architectures of multicore CPU, GPUs and HPC clusters
    - Including the key concepts in parallel computer architectures, e.g. shared memory system, distributed system, NUMA and cache coherence, interconnection
  - Understand principles for parallel and concurrent program design, e.g. decomposition of works, task and data parallelism, processor mapping, mutual exclusion, locks.
- Develop skills writing and analyzing parallel programs
  - Write parallel program using OpenMP, CUDA, and MPI programming models.
  - Perform analysis of parallel program problem.

Textbooks



- **Required:** Introduction to Parallel Computing (2nd Edition), PDF, Amazon, cover theory, MPI and OpenMP introduction, by Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar, Addison-Wesley, 2003
- **Recommended**: John Cheng, Max Grossman, and Ty McKercher, Professional CUDA C Programming, 1st Edition 2014, PDF, Amazon.
- **Reference book for OpenMP**: Barbara Chapman, Gabriele Jost, and Ruud van der Pas, Using OpenMP: Portable Shared Memory Parallel Programming, 2007, PDF, Amazon.
- **Reference book for MPI**: Choose from Recommended Books for MPI

- Lots of materials on Internet.
  - On the website, there is a "Resources" section that provides web page links, documents, and other materials for this course

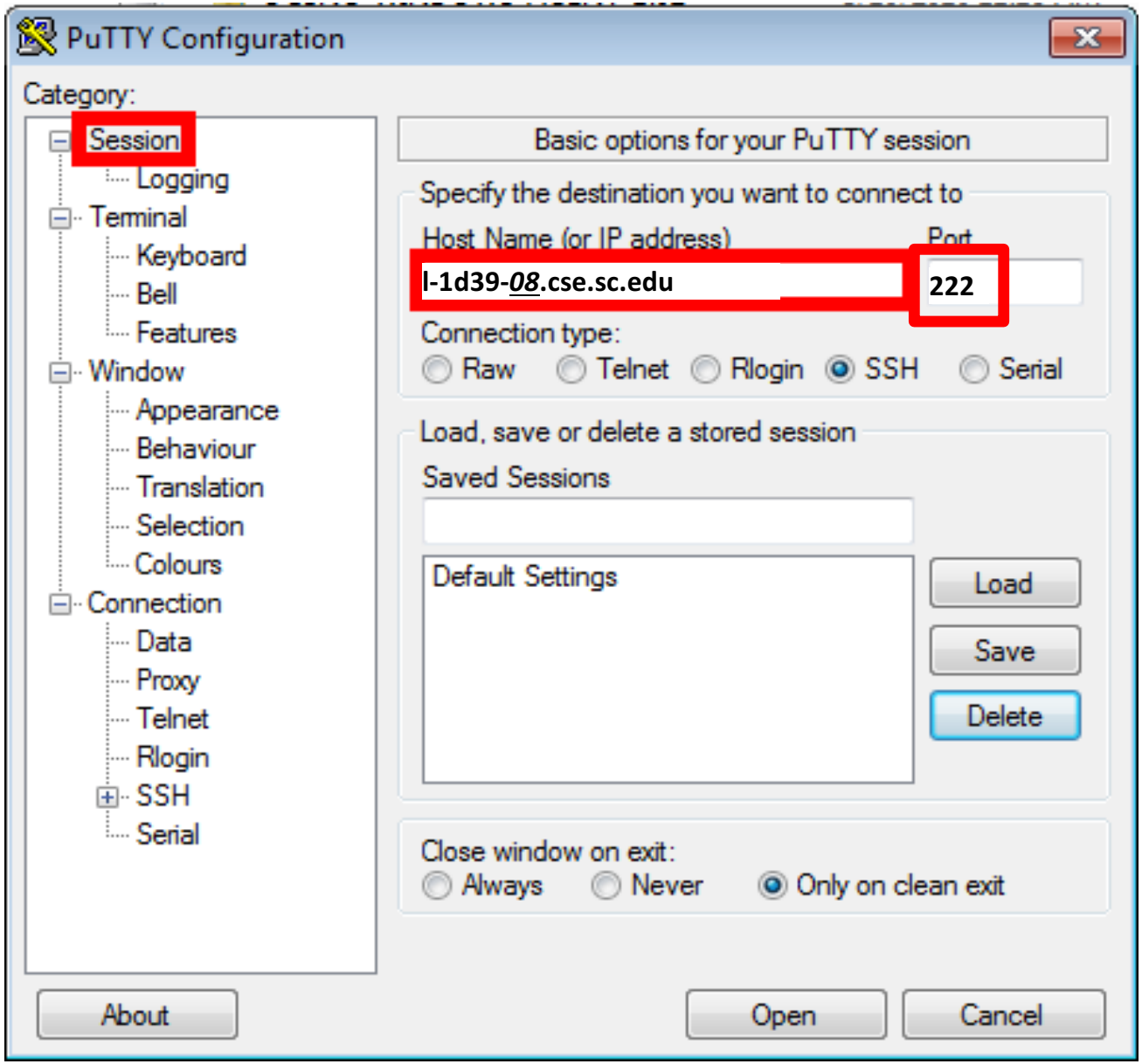
Homeworks and Exams

- **Four homeworks**: practice programming skills
  - Require both good and correct programming
    - Write organized program that is easy to read
  - Report and discuss your findings in report
    - Writing good document
  - **60% Total (10% + 10% + 20% + 20%)**
- **Exams: Test fundamentals**
  - ***Close/Open book (?)***
  - **40% Total**
- **Midterm: 15%, March 7th Wednesday during class**
  - **The week before spring break.**
- **Final Exam: 25%, May 2nd Wednesday, 9:00AM - 11:30AM**

Machine for Development for OpenMP and MPI

- Linux machines in Swearingen 1D39 and 3D22
  - All CSCE students by default have access to these machine using their standard login credentials
    - Let me know if you, CSCE or not, cannot access
  - Remote access is also available via SSH over port 222. Naming schema is as follows:
    - l-1d39-*01*.cse.sc.edu through l-1d39-*26*.cse.sc.edu
    - l-3d22-*01*.cse.sc.edu through l-3d22-*20*.cse.sc.edu
- Restricted to 2GB of data in their home folder (~/).
  - For more space, create a directory in /scratch on the login machine, however that data is not shared and it will only be available on that specific machine.

Putty SSH Connection on Windows



SSH Connection from Linux/Mac OS X Terminal

```
yanyh@cocsce-l1d39-15:~/opencv$ exit
logout
Connection to l-1d39-15.cse.sc.edu closed.
MacBook-Pro-7:yanyh yanyh$ ssh -p 222 l-1d39-15.cse.sc.edu -lyanyh -X
**************************************************************************
*                                                                        *
* This system is for the use of authorized users only.  Usage of this system *
* may be monitored and recorded by system personnel.                     *
*                                                                        *
* Anyone using this system expressly consents to such monitoring and is  *
* advised that if such monitoring reveals possible evidence of criminal  *
* activity, system personnel may provide the evidence from such monitoring *
* to law enforcement officials.                                          *
*                                                                        *
**************************************************************************
Password:
/usr/bin/xauth:  file /acct/yanyh/.Xauthority does not exist
Duo two-factor login for yanyh

Enter a passcode or select one of the following options:

 1. Duo Push to XXX-XXX-5878
 2. Phone call to XXX-XXX-5878
 3. SMS passcodes to XXX-XXX-5878

Passcode or option (1-3): 1

Pushed a login request to your device...
Success. Logging you in...
yanyh@cocsce-l1d39-15:~$ ls
```

-X for enabling X-windows forwarding so you can use the graphics display on your computer. For Mac OS X, you need have X server software installed, e.g. Xquartz(https://www.xquartz.org/) is the one I use.

Try in The Lab and From Remote

- Bring your laptop

Topics

- Introduction
- Programming on shared memory system (Chapter 7)
  - **OpenMP**
  - **PThread, mutual exclusion, locks, synchronizations**
  - ***Cilk/Cilkplus(?)***
- Principles of parallel algorithm design (Chapter 3)
- Analysis of parallel program executions (Chapter 5)
  - **Performance Metrics for Parallel Systems**
    - **Execution Time, Overhead, Speedup, Efficiency, Cost**
  - **Scalability of Parallel Systems**
  - **Use of performance tools**

Topics

- Programming on large scale systems (Chapter 6)
  - **MPI (point to point and collectives)**
  - Introduction to PGAS languages, UPC and Chapel (?)
- Parallel architectures and hardware
  - **Parallel computer architectures**
  - **Memory hierarchy and cache coherency**
- Manycore GPU architectures and programming
  - **GPUs architectures**
  - **CUDA programming**
  - Introduction to offloading model in OpenMP(?)
- Parallel algorithms (Chapter 8,9 &10)
  - ***Dense linear algebra, stencil and image processing***

Prerequisites

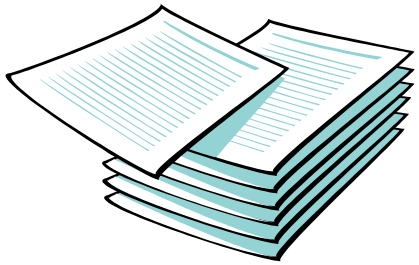
- Good reasoning and analytical skills
- Familiarity with and Skills of C/C++ programming
  - macro, pointer, array, struct, union, function pointer, etc.
- Familiarity with Linux environment
  - SSH, Linux commands, vim/Emacs editor
- Basic knowledge of computer architecture and data structures
  - Memory hierarchy, cache, virtual address
  - Array and link-list
- Talk with me if you have concern
- **Turn in the survey**

Introduction: What is and why Parallel Computing

An Example: Grading

15 questions

300 exams

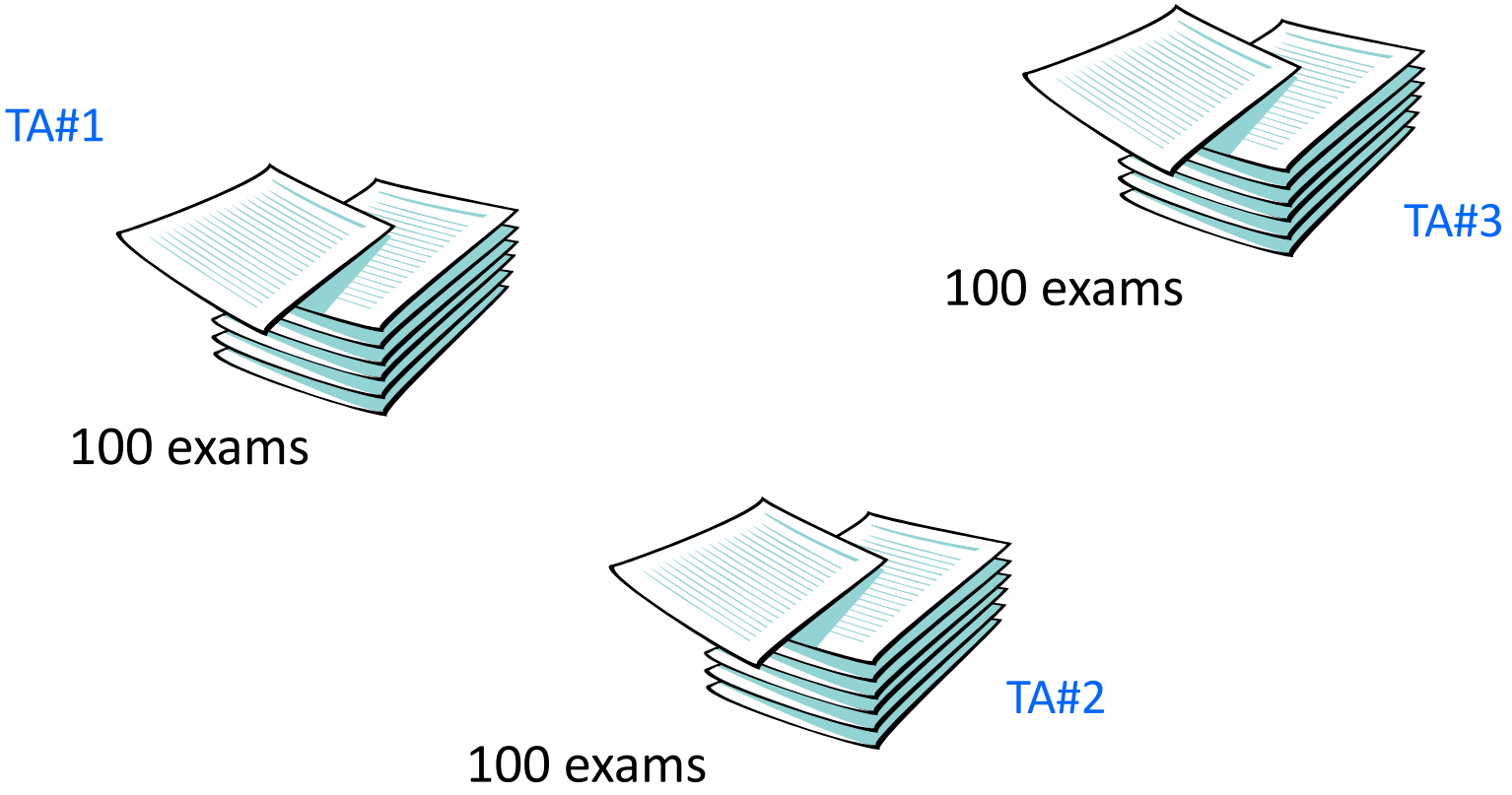


Three Teaching Assistants

- To grade 300 copies of exams, each has 15 questions

Division of Work – Data Parallelism

- Each does the same type of work (task), but working on different sheet (data)



Division of Work – Task Parallelism

- Each does different type of work (task), but working on same sheets (data)

TA#1

Questions 1 - 5

TA#3

Questions 11 - 15

TA#2

Questions 6 - 10

Summary

- Data: 300 copies of exam
- Task: grade total 300*15 questions
- Data parallelism
  - **Distributed 300 copies to three TAs**
  - **They work independently**
- Task Parallelism
  - **Distributed 300 copies to three TAs**
  - **Each grades 5 questions of 100 copies**
  - **Exchange copies**
  - **Grade 5 questions again**
  - **Exchange copies**
  - **Grade 5 questions**
- The three TAs can do in parallel, we can achieve 3 time speedup theoretically

Which approach could be faster!

Challenges

- Are the three TAs grading in the same performance?
  - One CPU may be slower than the other
  - They may not work on grading the same time
- How the TAs communicate?
  - Are they sit on the same table? Or each take copies and grade from home? How they share intermediate results (task parallelism)
- Where the solutions are stored so they can refer to when grading
  - Remember answers to 5 questions vs to 15 questions
    - Cache and Memory issues

What is Parallel Computing?

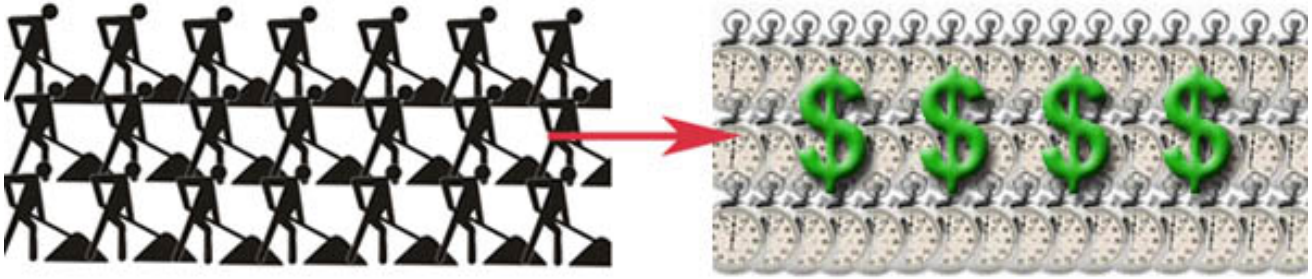
- **A form of computation*:**
  - **Large problems divided into smaller ones**
  - **Smaller ones are carried out and solved simultaneously**

- Uses more than one CPUs or cores concurrently for one program
  - Not conventional time-sharing: multiple programs switch between each other on one CPU
  - Or multiple programs each on a CPU and not interacting
- Serial processing
  - Some programs, or part of a program are inherently serial
  - Most of our programs and desktop applications

*http://en.wikipedia.org/wiki/Parallel_computing

Why Parallel Computing?

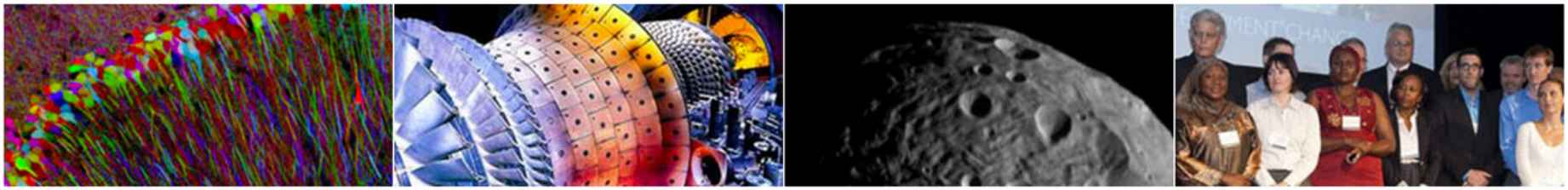
- **Save time (execution time) and money!**
  - **Parallel program can run faster if running concurrently instead of sequentially.**



Picture from: Intro to Parallel Computing: https://computing.llnl.gov/tutorials/parallel_comp

- **Solve larger and more complex problems!**
  - **Utilize more computational resources**

Current Grand Challenges



NIH, DARPA, and NSF's BRAIN Initiative, to revolutionize our understanding of the human mind and

DOE's SunShot Grand Challenge, to make solar energy cost competitive with coal by the end of the decade, and EV

NASA's Asteroid Grand Challenge, to find all asteroid threats to human populations and know what to do about

USAID's Grand Challenges for Development, including **Saving Lives at Birth** that catalyzes groundbreaking

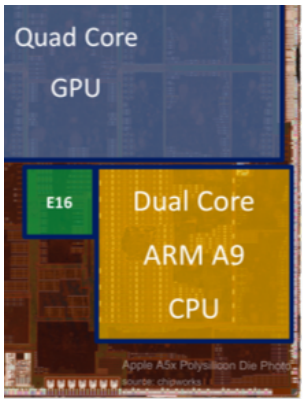
From "21st Century Grand Challenges | The White House", http://www.whitehouse.gov/administration/eop/ostp/grand-challenges
Grand challenges: http://en.wikipedia.org/wiki/Grand_Challenges

High Performance Computing (HPC) and Parallel Computing

- HPC is what really needed *
  - Parallel computing is so far the only way to get there!!
- Parallel computing makes sense!
- Applications that require HPC
  - Many problem domains are naturally parallelizable
  - Data cannot fit in memory of one machine
- Computer systems
  - Physics limitation: has to build it parallel
  - Parallel systems are widely accessible
    - Smartphone has 2 to 4 cores + GPU now

We will discuss e

the two aspect t



*What is HPC: http://insidehpc.com/hpc-basic-training/what-is-hpc/
Supercomputer: http://en.wikipedia.org/wiki/Supercomputer
TOP500 (500 most powerful computer systems in the world): http://en.wikipedia.org/wiki/TOP500, http://top500.org/
HPC matter: http://sc14.supercomputing.org/media/social-media

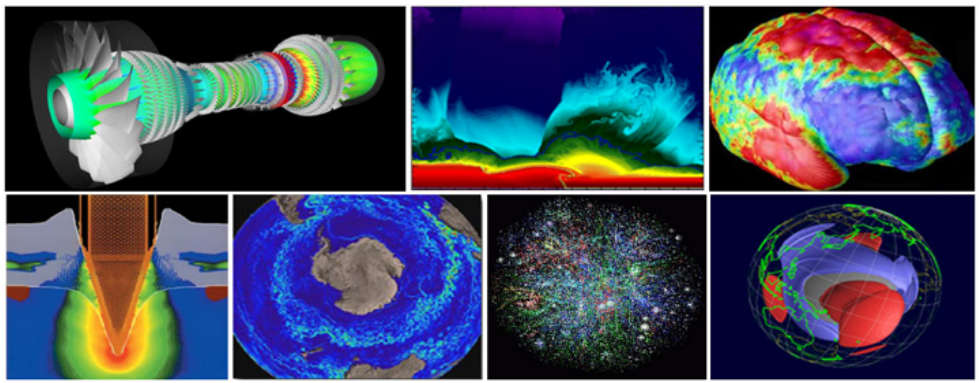
Simulation: The *Third* Pillar of Science

- **Traditional scientific and engineering paradigm:**
  1) **Do theory or paper design.**
  2) **Perform experiments or build system.**
- Limitations of experiments:
  - Too difficult -- build large wind tunnels.
  - Too expensive -- build a throw-away passenger jet.
  - Too slow -- wait for climate or galactic evolution.
  - Too dangerous -- weapons, drug design, climate experimentation.
- **Computational science paradigm:**
  3) **Use high performance computer systems to simulate the phenomenon**
     - **Base on known physical laws and efficient numerical methods.**

From slides of Kathy Yelic's 2007 course at Berkeley: http://www.cs.berkeley.edu/~yelick/cs267_sp07/

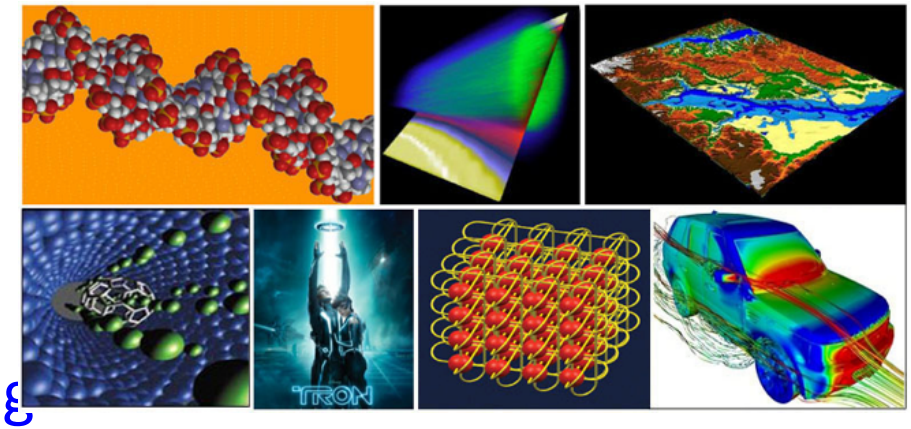
Applications: Science and Engineering

- Model many difficult problems by parallel computing
  - Atmosphere, Earth, Environment
  - Physics - applied, nuclear, particle, condensed matter, high pressure, fusion, photonics
  - Bioscience, Biotechnology, Genetics
  - Chemistry, Molecular Sciences
  - Geology, Seismology
  - Mechanical Engineering - from prosthetics to spacecraft
  - Electrical Engineering, Circuit Design, Microelectronics
  - Computer Science, Mathematics
  - Defense, Weapons



Applications: Industrial and Commercial

- Processing large amounts of data in sophisticated ways
  - Databases, data mining
  - Oil exploration
  - Medical imaging and diagnosis
  - Pharmaceutical design
  - Financial and economic modeling
  - Management of national and multi-national corporations
  - Advanced graphics and virtual reality, particularly in the entertainment industry
  - Networked video and multi-media technologies
  - Collaborative work environments
  - Web search engines, web based business services

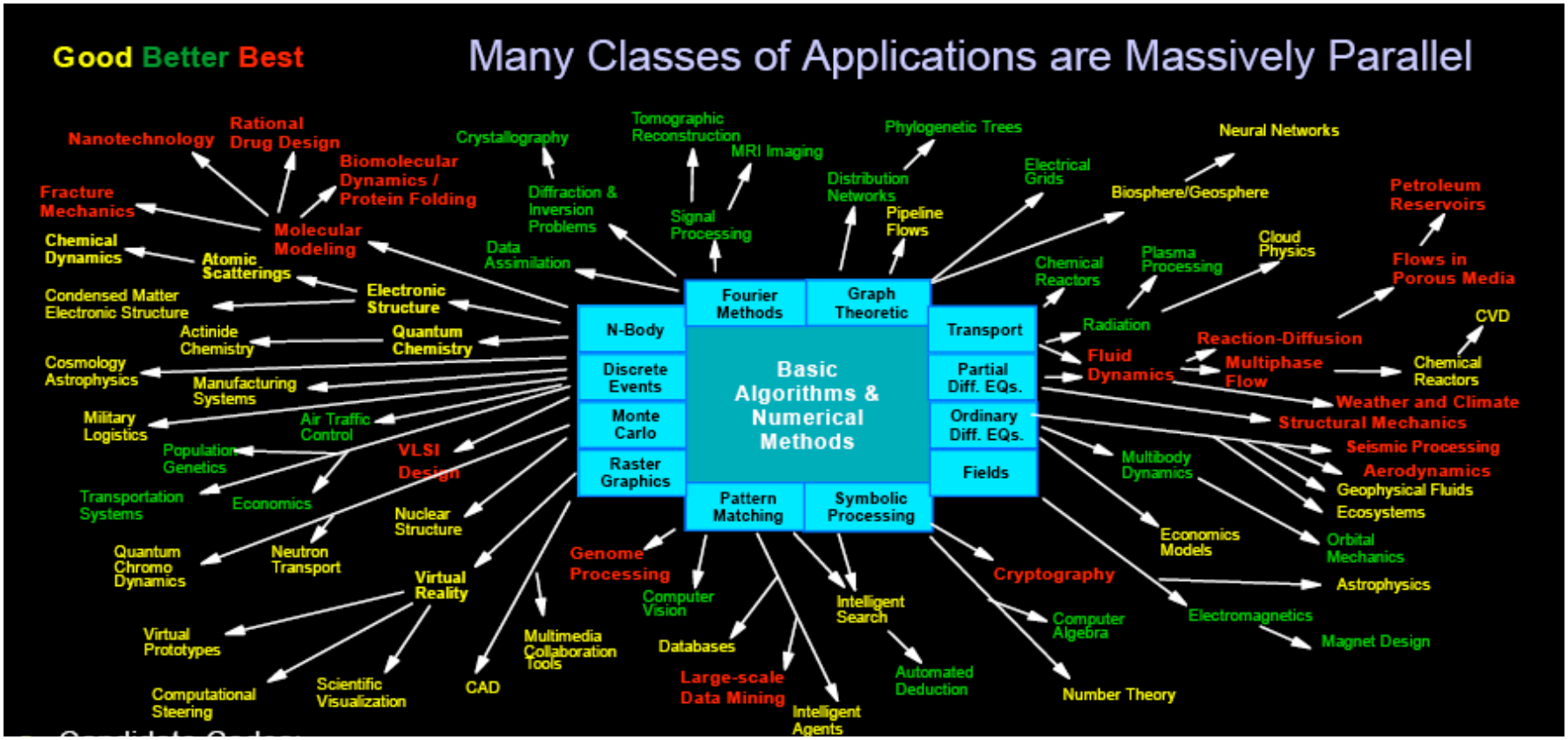


Economic Impact of HPC

- Airlines:
  - System-wide logistics optimization systems on parallel systems.
  - Savings: approx. $100 million per airline per year.
- Automotive design:
  - Major automotive companies use large systems (500+ CPUs) for:
    - CAD-CAM, crash testing, structural integrity and aerodynamics.
    - One company has 500+ CPU parallel system.
  - Savings: approx. $1 billion per company per year.
- Semiconductor industry:
  - Semiconductor firms use large systems (500+ CPUs) for
    - device electronics simulation and logic validation
  - Savings: approx. $1 billion per company per year.
- Securities industry:
  - Savings: approx. $15 billion per year for U.S. home mortgages.

From slides of Kathy Yelic's 2007 course at Berkeley: http://www.cs.berkeley.edu/~yelick/cs267_sp07/

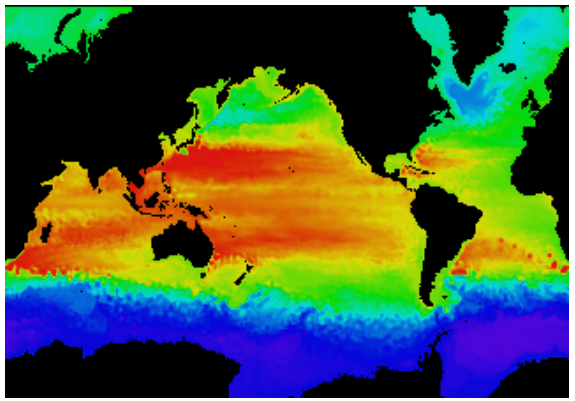
Inherent Parallelism of Applications

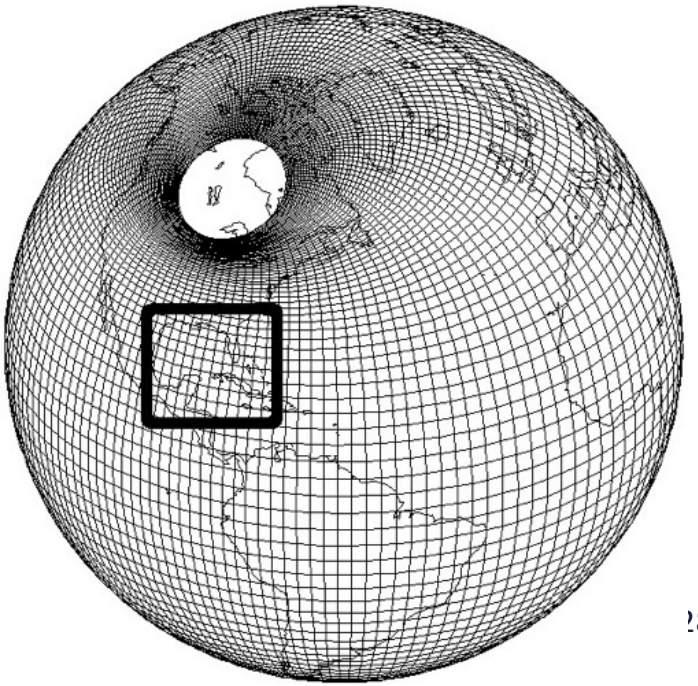


- **Example: weather prediction and global climate modeling**

Global Climate Modeling Problem

- **Problem is to compute:**
  - **f(latitude, longitude, elevation, time) →**
    temperature, pressure, humidity, wind velocity
- Approach:
  - *Discretize* the domain, e.g., a measurement point every 10 km
  - Devise an algorithm to predict weather at time t+dt given t
- Uses:
  - Predict major events, e.g., El Nino
  - Air quality forecasting





The Rise of Multicore Processors

Recent Multicore Processors

- Sept 13: Intel Ivy Bridge-EP Xeon E5-2695 v2
  - — 12 cores; 2-way SMT; 30MB cache
- March 13: SPARC T5
  - — 16 cores; 8-way fine-grain MT per core
- May 12: AMD Trinity
  - — 4 CPU cores; 384 graphics cores
- Nov 12: Intel Xeon Phi coprocessor
  - — ~60 cores
- Feb 12: Blue Gene/Q
  - — 17 cores; 4-way SMT
- Q4 11: Intel Ivy Bridge
  - — 4 cores; 2 way SMT;
- November 11: AMD Interlagos
  - — 16 cores
- Jan 10: IBM Power 7
  - — 8 cores; 4-way SMT; 32MB shared cache
- Tilera TilePro64

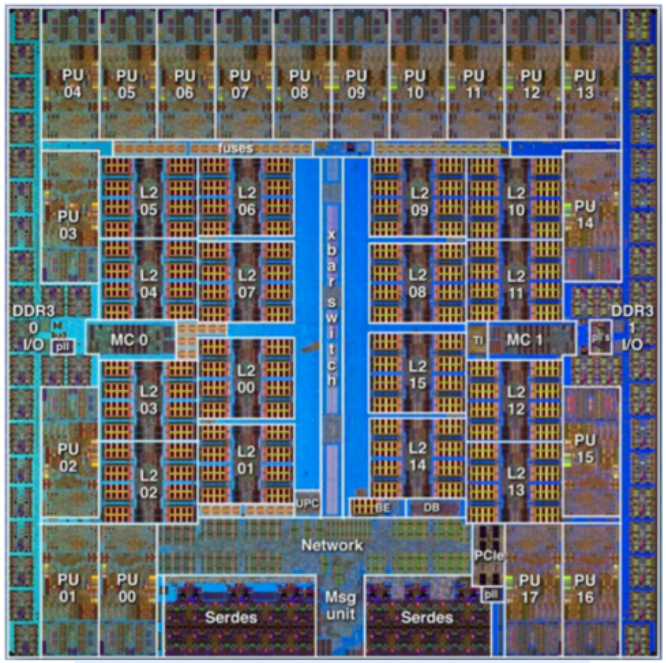
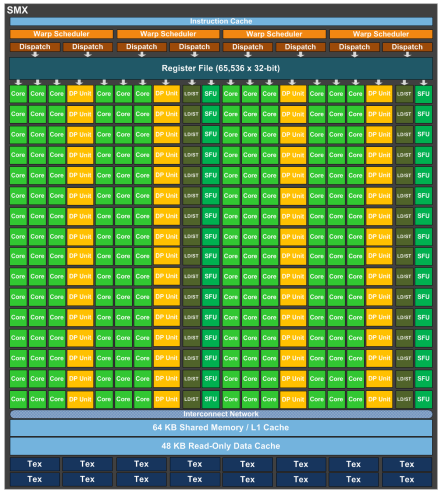


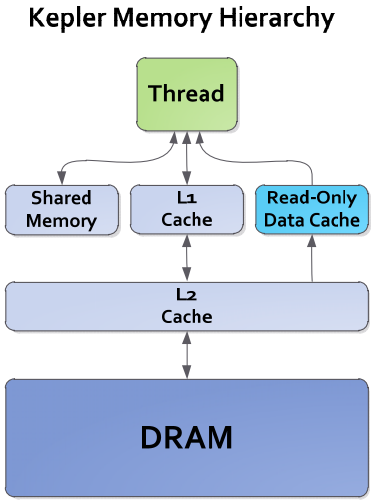
Figure credit: Ruud Haring, Blue Gene/Q compute chip, Hot Chips 23, August, 2011.

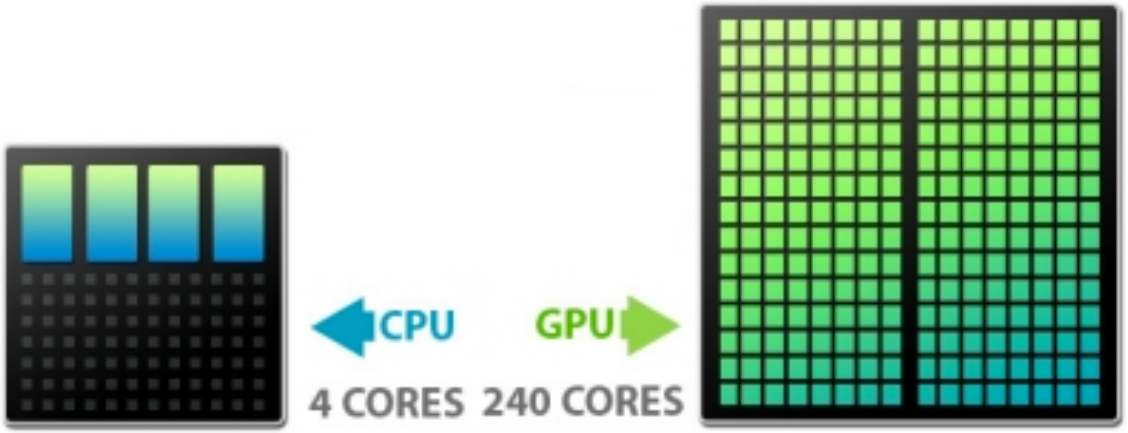
Recent Manycore GPU processors

- ~3k cores



SMX: 192 single-precision CUDA cores, 64 double-precision units, 32 special function units (SFU), and 32 load/store units (LD/ST).





Units of Measure in HPC

- **Flop**: floating point operation (*, /, +, -, etc)
- **Flop/s**: floating point operations per second, written also as **FLOPS**
- **Bytes**: size of data
  - **A double precision floating point number is 8 bytes**
- Typical sizes are millions, billions, trillions…
  - Mega Mflop/s = 10^6 flop/sec Mzbyte = 2^{20} = 1048576 = ~10^6 bytes
  - Giga Gflop/s = 10^9 flop/sec Gbyte = 2^{30} = ~10^9 bytes
  - Tera Tflop/s = 10^{12} flop/sec Tbyte = 2^{40} = ~10^{12} bytes
  - **Peta Pflop/s = 10^{15} flop/sec Pbyte = 2^{50} = ~10^{15} bytes**
  - Exa Eflop/s = 10^{18} flop/sec Ebyte = 2^{60} = ~10^{18} bytes
  - Zetta Zflop/s = 10^{21} flop/sec Zbyte = 2^{70} = ~10^{21} bytes

- www.top500.org for the units of the fastest machines measured using High Performance LINPACK (HPL) Benchmark
  - The fastest: Sunway TaihuLight, ~93 petaflop/s
  - The third (fastest in US): DoE ORNL Titan, 17.59 petaflop/s

How to Measure and Calculate Performance (FLOPS)

```
    elapsed = read_timer();
    REAL result = sum(N, X, a);
    elapsed = (read_timer() - elapsed);

    double elapsed_2 = read_timer();
    result = sumaxpy(N, X, Y, a);
    elapsed_2 = (read_timer() - elapsed_2);

    /* you should add the call to each function and time the execution */
    printf("================================================================================");
    printf("\tSum %d numbers\n", N);
    printf("--------------------------------------------------------------------------------");
    printf("Performance:\t\tRuntime (ms)\t MFLOPS \n");
    printf("--------------------------------------------------------------------------------");
    printf("Sum:\t\t\t%4f\t%4f\n", elapsed * 1.0e3, 2*N / (1.0e6 * elapsed));
    printf("SumAXPY:\t\t\t%4f\t%4f\n", elapsed_2 * 1.0e3, 3*N / (1.0e6 * elapsed_2));
    return 0;
}

REAL sum(int N, REAL X[], REAL a) {
    int i;
    REAL result = 0.0;
    for (i = 0; i < N; ++i)
        result += a * X[i];
    return result;
}

/*
 * sum: a*X[]+Y[]
 */
REAL sumaxpy(int N, REAL X[], REAL Y[], REAL a) {
    int i;
    REAL result = 0.0;
    for (i = 0; i < N; ++i)
        result += a * X[i] + Y[i];
    return result;
}
```

https://passlab.github.io/CSCE569/resources/sum.c

- **Calculate # FLOPs (2*N or 3*N)**
  - **Check the loop count (N) and FLOPs per loop iteration (2 or 3).**
- **Measure time to compute using timer**
  - **elapsed and elapsed_2 are in second**
- **FLOPS = # FLOPs/Time**
  - **MFLOPS in the example**

High Performance LINPACK (HPL) Benchmark Performance (Rmax) in Top500

- ***Measured*** **using the High Performance LINPACK (HPC) Benchmark that solves a dense system of linear equations**
  → Ranking the machines
  - **Ax = b**
  - **https://www.top500.org/project/linpack/**
  - **https://en.wikipedia.org/wiki/LINPACK_benchmarks**

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272

Top500 (www.top500.org), Nov 2017

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
4	**Gyoukou** - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz , ExaScaler Japan Agency for Marine-Earth Science and Technology Japan	19,860,000	19,135.8	28,192.0	1,350
5	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
6	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890
7	**Trinity** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect	979,968	14,137.3	43,902.6	3,844

HPC Peak Performance (Rpeak) Calculation

- **Node performance in Gflop/s = (CPU speed in GHz) x (number of CPU cores) x (CPU instruction per cycle) x (number of CPUs per node).**
  - **CPU instructions per cycle (IPC) = #Flops per cycle**
    - **Because pipelined CPU can do one instruction per cycle**
    - **4 or 8 for most CPU (Intel or AMD)**
  - **http://www.calcverter.com/calculation/CPU-peak-theoretical-performance.php**

- **HPC Peak (Rpeak) = # nodes * Node Performance in GFlops**

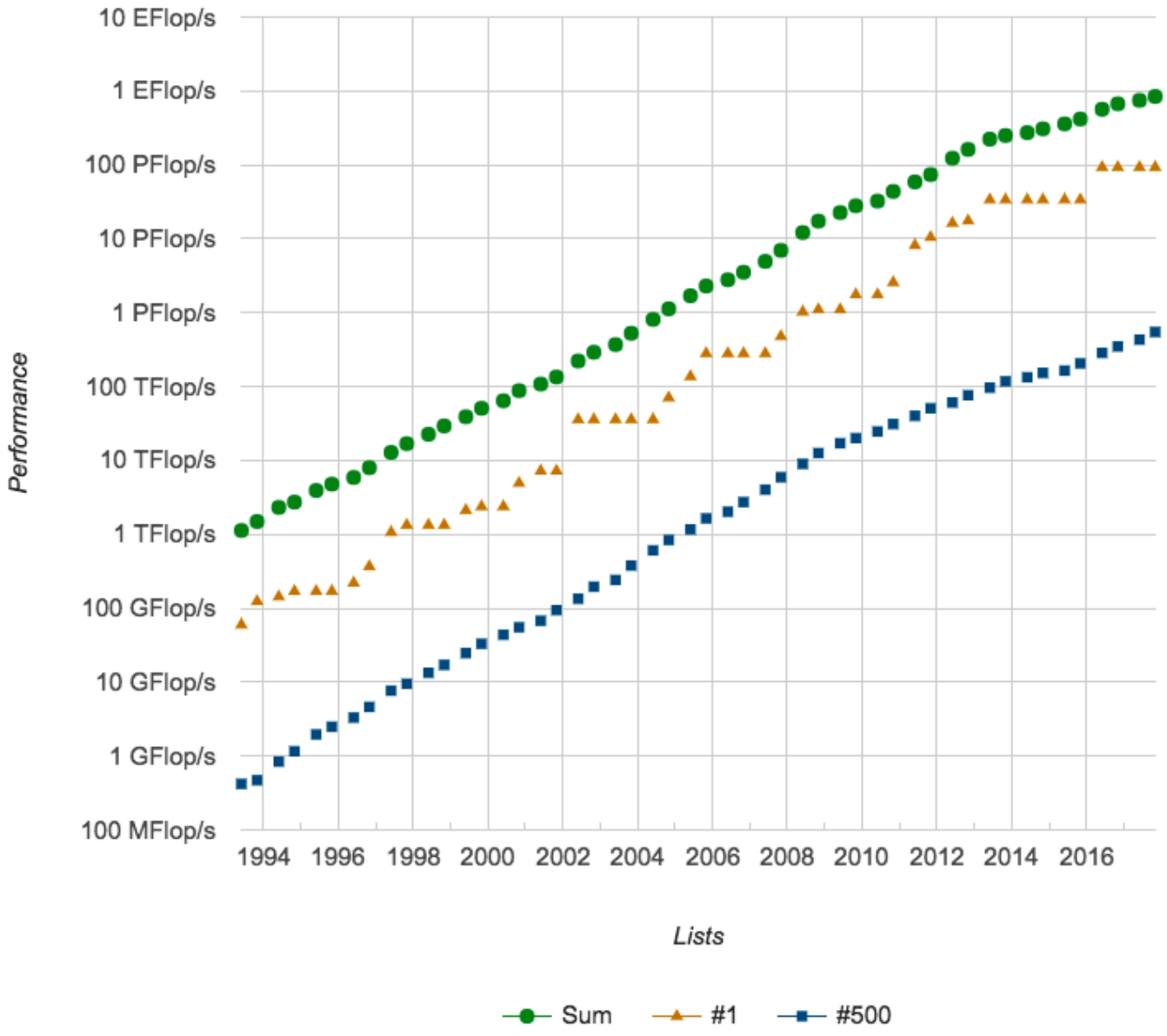
CPU Peak Performance Example

- Intel X5600 series CPUs and AMD 6100/6200/6300 series CPUs have 4 instructions per cycle
  Intel E5-2600 series CPUs have 8 instructions per cycle

- Example 1: Dual-CPU server based on Intel X5675 (3.06GHz 6-cores) CPUs:
  - 3.06 x 6 x 4 x 2 = 144.88 GFLOPS
- Example 2: Dual-CPU server based on Intel E5-2670 (2.6GHz 8-cores) CPUs:
  - 2.6 x 8 x 8 x 2 = 332.8 GFLOPS
  - With 8 nodes: 332.8 GFLOPS x 8 = 2,442.4 GFLOPS = 2.44 TFLOPS
- Example 3: Dual-CPU server based on AMD 6176 (2.3GHz 12-cores) CPUs:
  - 2.3 x 12 x 4 x 2 = 220.8 GFLOPS
- Example 4: Dual-CPU server based on AMD 6274 (2.2GHz 16-cores) CPUs:
  - 2.2 x 16 x 4 x 2 = 281.6 GFLOPS

https://saiclearning.wordpress.com/2014/04/08/how-to-calculate-peak-theoretical-performance-of-a-cpu-based-hpc-system/

Performance (HPL) Development Over Years of Top500 Machines



4 Kinds of Ranking of HPC/Supercomputers

1. **Top500: according to the *Measured* *High Performance LINPACK (HPL) Benchmark performance***
   - ***Not Peak performance, Not other applications***

2. **Ranking according to *HPCG* benchmark performance**
3. **Graph500 Ranking according to graph processing capability**
   - **Shortest Path and Breadth First Search**
   - **https://graph500.org**
4. **Green500 Ranking according to *Power efficiency (GFLOPS/Watts)***
   - **https://www.top500.org/green500/**
   - **Generate sublist in the following slides from https://www.top500.org/statistics/sublist/**

HPCG Ranking

- **HPCG: High Performance Conjugate Gradients (HPCG) Benchmark (http://www.hpcg-benchmark.org/)**

Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	HPCG (TFlop/s)
1	10	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect , Fujitsu RIKEN Advanced Institute for Computational Science (AICS) Japan	705,024	10,510.0	11,280.4	602.736
2	2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	580.109
3	7	**Trinity** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/NNSA/LANL/SNL United States	979,968	14,137.3	43,902.6	546.124
4	3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	486.398
5	1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	480.8
6	9	**Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path , Fujitsu	556,104	13,554.6	24,913.5	385.479

Graph500 (https://graph500.org)

- **Ranking according to the capability of processing large-scale graph (Shortest Path and Breadth First Search)**

Top Ten from November 2017 BFS

RANK	PREVIOUS RANK	MACHINE	VENDOR	TYPE	NETWORK	INSTALLATION SITE	LOCATION	COUNTRY	YEAR	APPLICATION
1	1	K computer	Fujitsu	Custom	Tofu	RIKEN Advanced Institute for Computational Science (AICS)	Kobe Hyogo	Japan	2011	Various scie and instudri fields
2	2	Sunway TaihuLight	NRCPC	Sunway MPP	Sunway	National Supercomputing Center in Wuxi	Wuxi	China	2015	research
3	3	DOE/NNSA/LLNL Sequoia	IBM	BlueGene/Q Power BQC 16C 1.60 GHz	Custom	Lawrence Livermore National Laboratory	Livermore CA	USA	2012	Scientific Research
4	4	DOE/SC/Argonne National Laboratory Mira	IBM	BlueGene/Q Power BQC 16C 1.60 GHz	Custom	Argonne National Laboratory	Chicago IL	USA	2012	Scientific Research
5	5	JUQUEEN	IBM	BlueGene/Q Power BQC 16C 1.60 GHz	Custom	Forschungszentrum Juelich (FZJ)	Juelich	Germany	2012	Scientific Research
6	new	ALCF Mira - 8192	IBM	IBM -		Argonne National	Chicago IL	United	2012	Scientific

Green500: Power Efficiency (*GFLOPS/Watts*)

- **Power Efficiency = HPL Performance / Power**
  - **E.g. TaihuLight #1 of Top500: = 93,014.6 / 15,371 = 6.051 Gflops/watts)**
- **https://www.top500.org/green500/**

Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	Power (kW)	Power Efficiency (GFlops/watts)
20	1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	15,371	6.051

Green500: Power Efficiency (*GFLOPS/Watts*)

- https://www.top500.org/green500/

Rank	TOP500 Rank	System	Rmax (TFlop/s)	Power (kW)	Power Efficieny (GFlops/watts)
1	259	**Shoubu system B** - ZettaScaler-2.2, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 , PEZY Computing / Exascaler Inc. Advanced Center for Computing and Communication, RIKEN Japan	842.0	49.5	17.009
2	307	**Suiren2** - ZettaScaler-2.2, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 , PEZY Computing / Exascaler Inc. High Energy Accelerator Research Organization /KEK Japan	788.2	47.0	16.759
3	276	**Sakura** - ZettaScaler-2.2, Xeon E5-2618Lv3 8C 2.3GHz, Infiniband EDR, PEZY-SC2 , PEZY Computing / Exascaler Inc. PEZY Computing K.K. Japan	824.7	49.5	16.657
4	149	**DGX SaturnV Volta** - NVIDIA DGX-1 Volta36, Xeon E5-2698v4 20C 2.2GHz, Infiniband EDR, NVIDIA Tesla V100 , Nvidia NVIDIA Corporation United States	1,070.0	97	15.113
5	4	**Gyoukou** - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz , ExaScaler Japan Agency for Marine-Earth Science and Technology Japan	19,135.8	1,350.2	14.173
6	13	**TSUBAME3.0** - SGI ICE XA, IP139-SXM2, Xeon E5-2680v4 14C 2.4GHz,	8,125.0	792.1	13.704

Performance Efficiency

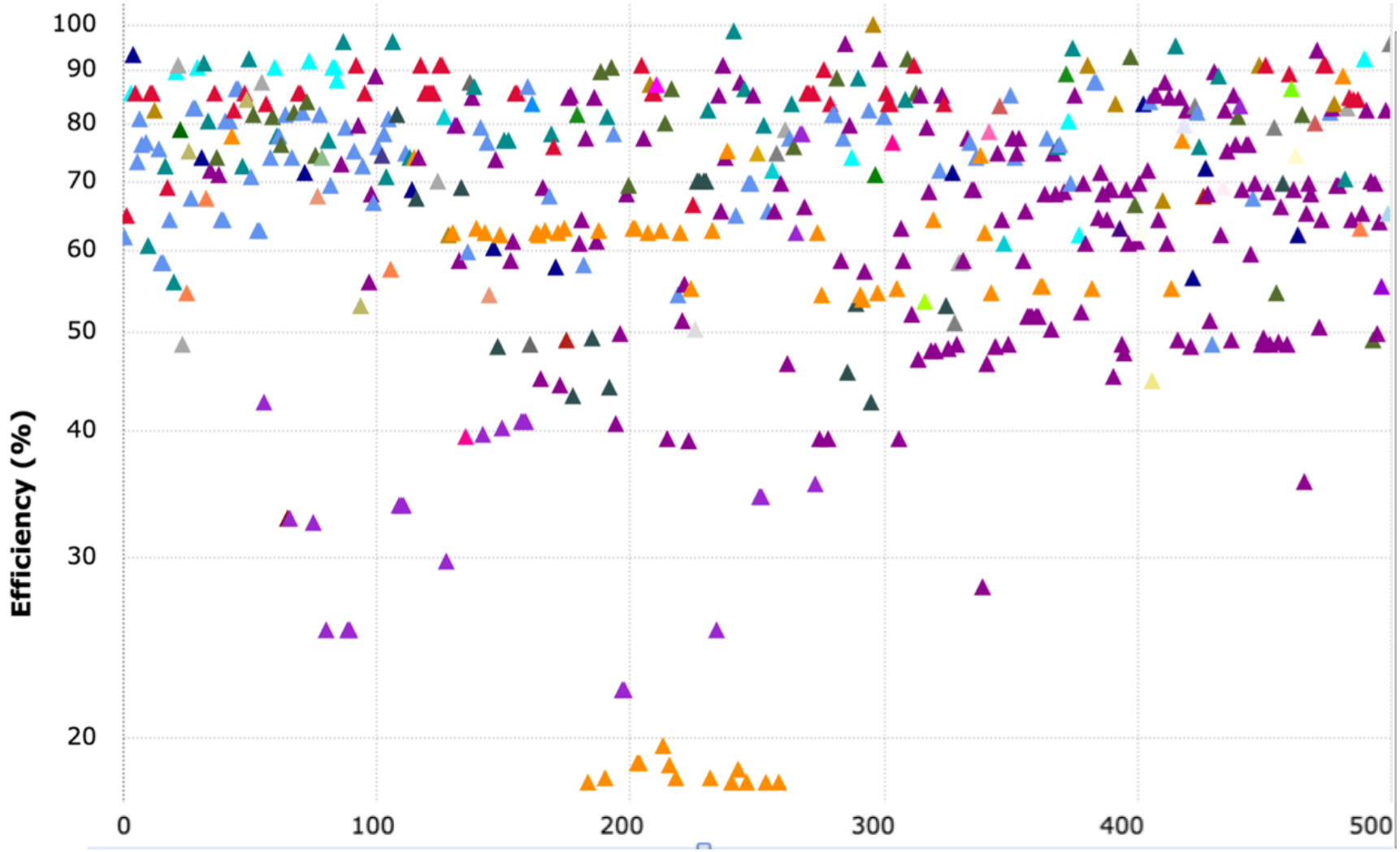
- **HPC Performance Efficiency = Actual *Measured* Performance GFLOPS / Theoretical Peak Performance GFLOPS**
  - **E.g. #1 in Top500**
    - **93,014.6/125,435.9 = 74.2%**

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9

https://www.penguincomputing.com/company/blog/calculate-hpc-efficiency/

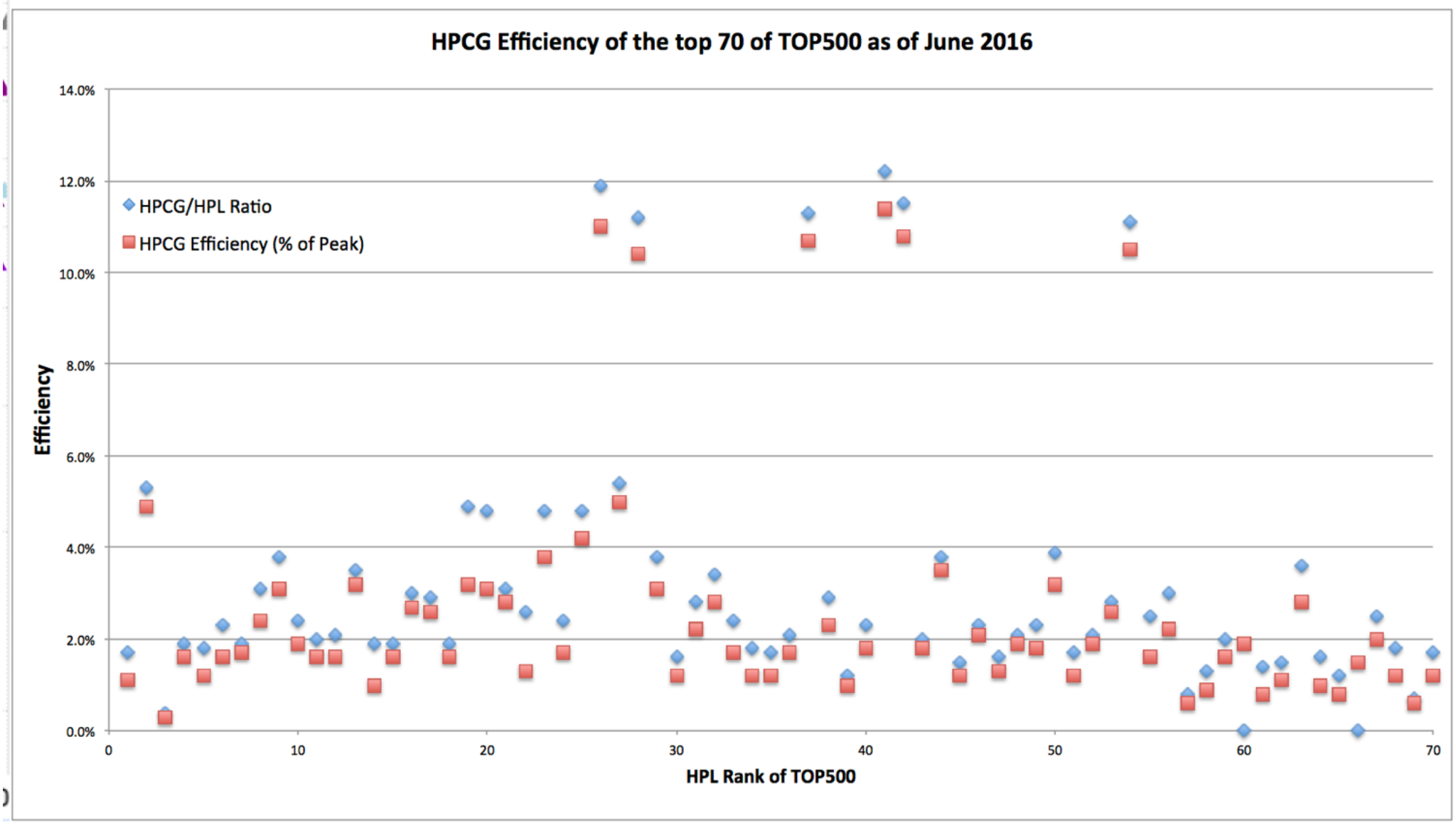
HPL Performance Efficiency of Top500 (2015 list)

- *Mostly 40% - 90% (ok)*



HPCG Efficiency of Top 70 of Top500 (2015 list)

- ***Mostly below 5% and only some around 10%***



Ranking Summary

- High Performance LINPACK (HPL) for Top500
  - Dense linear algebra (Ax = b), highly computation intensive
  - Rank Top500 for absolute computation capability

- HPCG: High Performance Conjugate Gradients (HPCG) Benchmark, HPL alternatives
  - Sparse Matrix-vector multiplication, balanced memory and computation intensity
  - Ranking machines with regards to the combination of computation and memory performance

- Graph500: Shortest Path and Breadth First Search
  - Ranking according to the capability of processing large-scale graph
  - Stressing network and memory systems

- Green500 of Top500 (HPL GFlops/watts)
  - Power efficiency

Why is parallel computing, namely multicore, manycore and clusters, the only way, so far, for high performance?

Semiconductor Trend: "Moore's Law"

Gordon Moore, Founder of Intel

- 1965: since the integrated circuit was invented, the number of transistors/inch^2 in these circuits roughly doubled every year; this trend would continue for the foreseeable future
- 1975: revised - circuit complexity doubles every two years

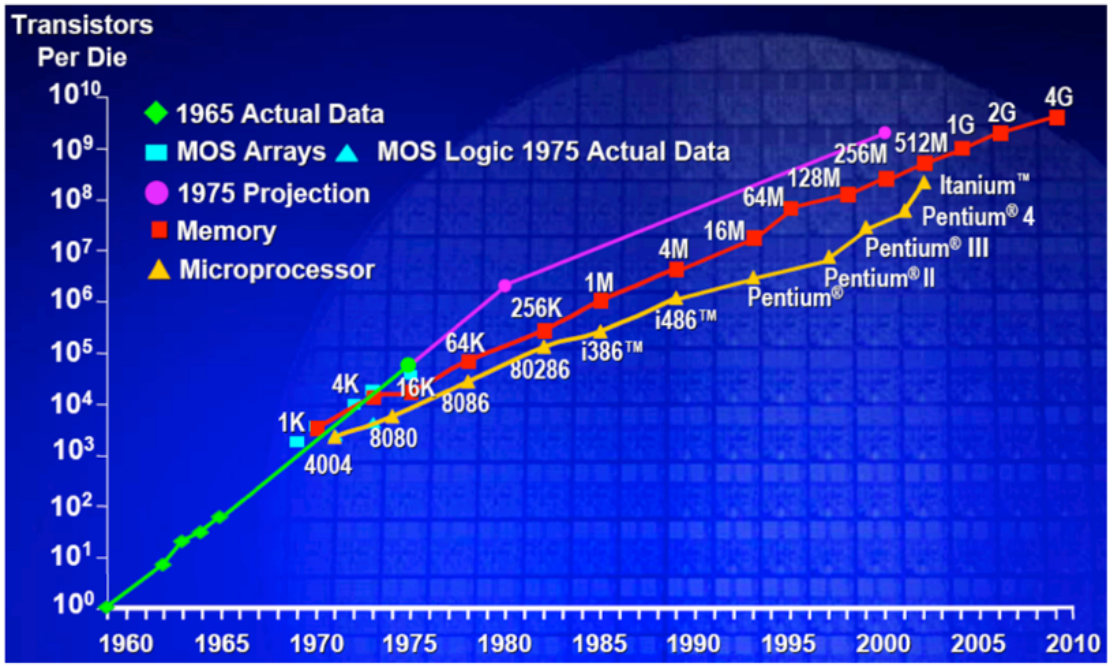
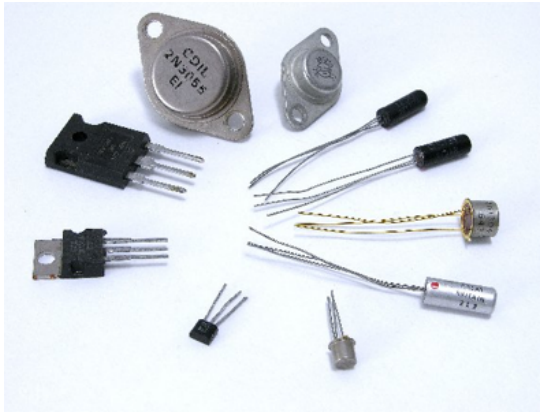
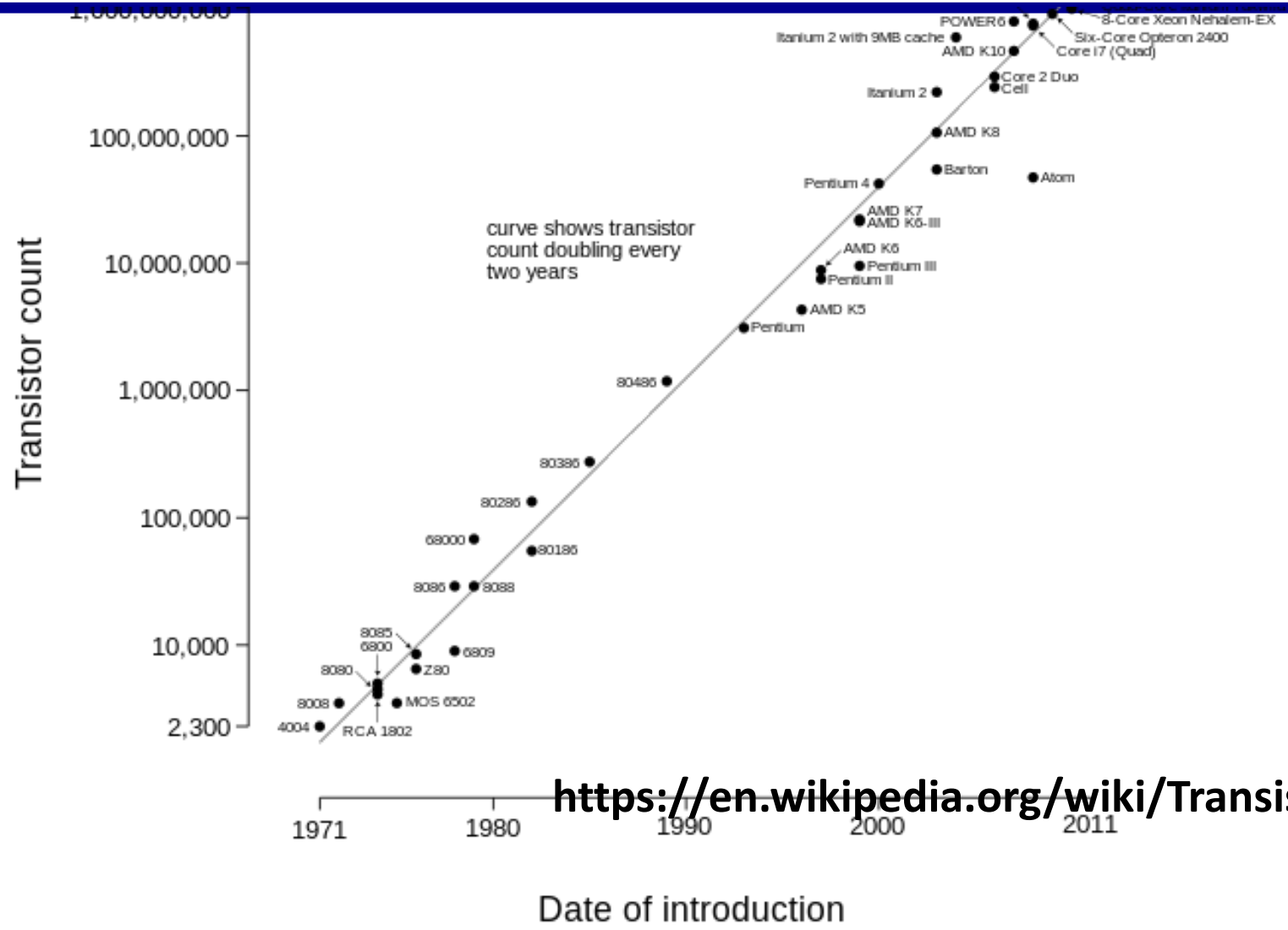


Image credit: Intel



Microprocessor Transistor Counts 1971-2011 & Moore's Law



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

Moore's Law Trends

- More transistors = ↑ opportunities for exploiting parallelism in the instruction level (ILP)
  - Pipeline, superscalar, VLIW (Very Long Instruction Word), SIMD (Single Instruction Multiple Data) or vector, speculation, branch prediction
- General path of scaling
  - Wider instruction issue, longer piepline
  - More speculation
  - More and larger registers and cache
- **Increasing circuit density ~= increasing frequency ~= increasing performance**
- Transparent to users
  - An easy job of getting better performance: buying faster processors (higher frequency)

- **We have enjoyed this free lunch for several decades, however …**

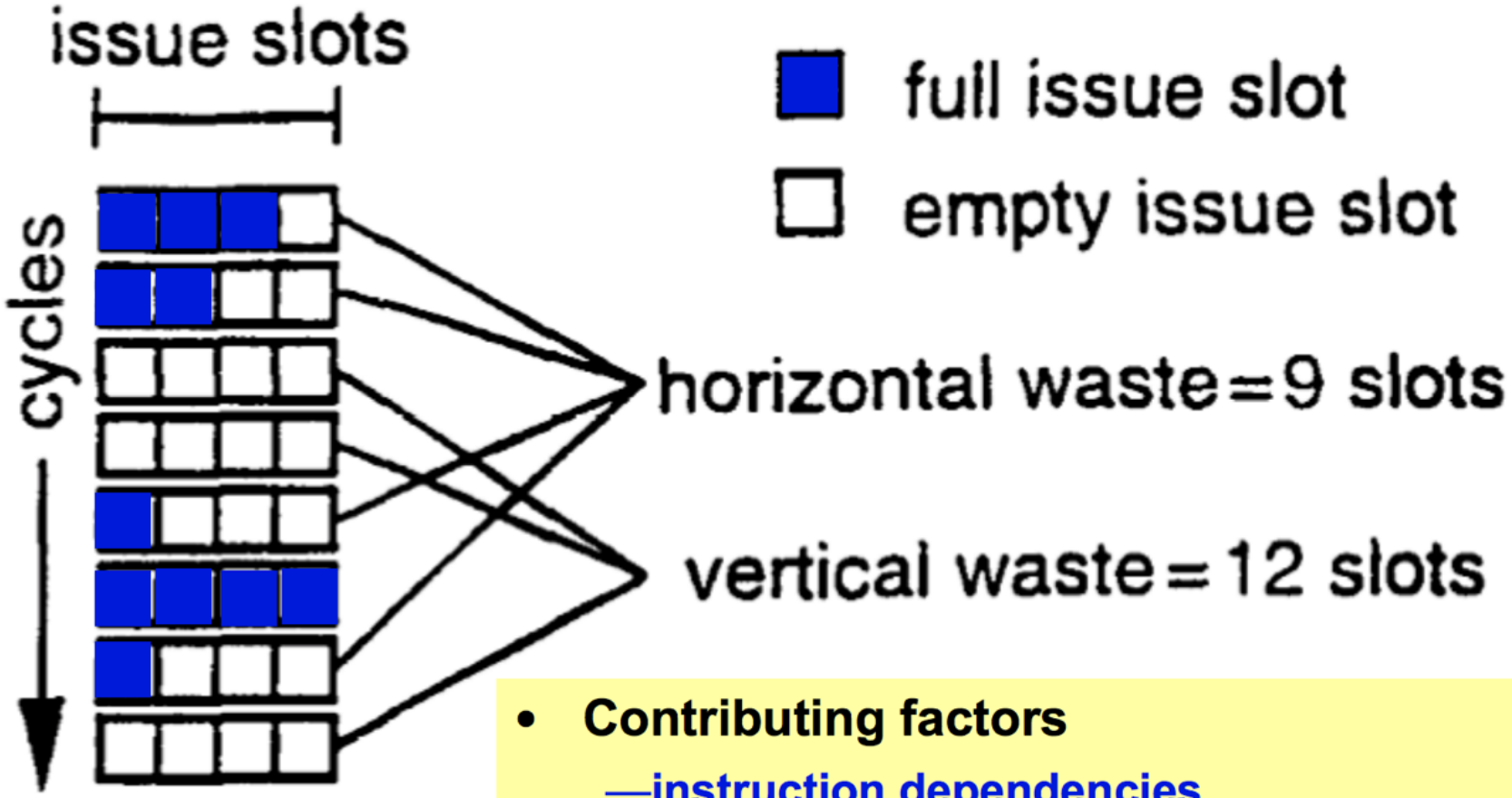
Problems of Traditional ILP Scaling

- Fundamental circuit limitations[1]
  - delays ⇑ as issue queues ⇑ and multi-port register files ⇑
  - increasing delays limit performance returns from wider issue
- Limited amount of instruction-level parallelism[1]
  - inefficient for codes with difficult-to-predict branches
- Power and heat stall clock frequencies

[1] The case for a single-chip multiprocessor, K. Olukotun, B. Nayfeh, L. Hammond, K. Wilson, and K. Chang, ASPLOS-VII, 1996.

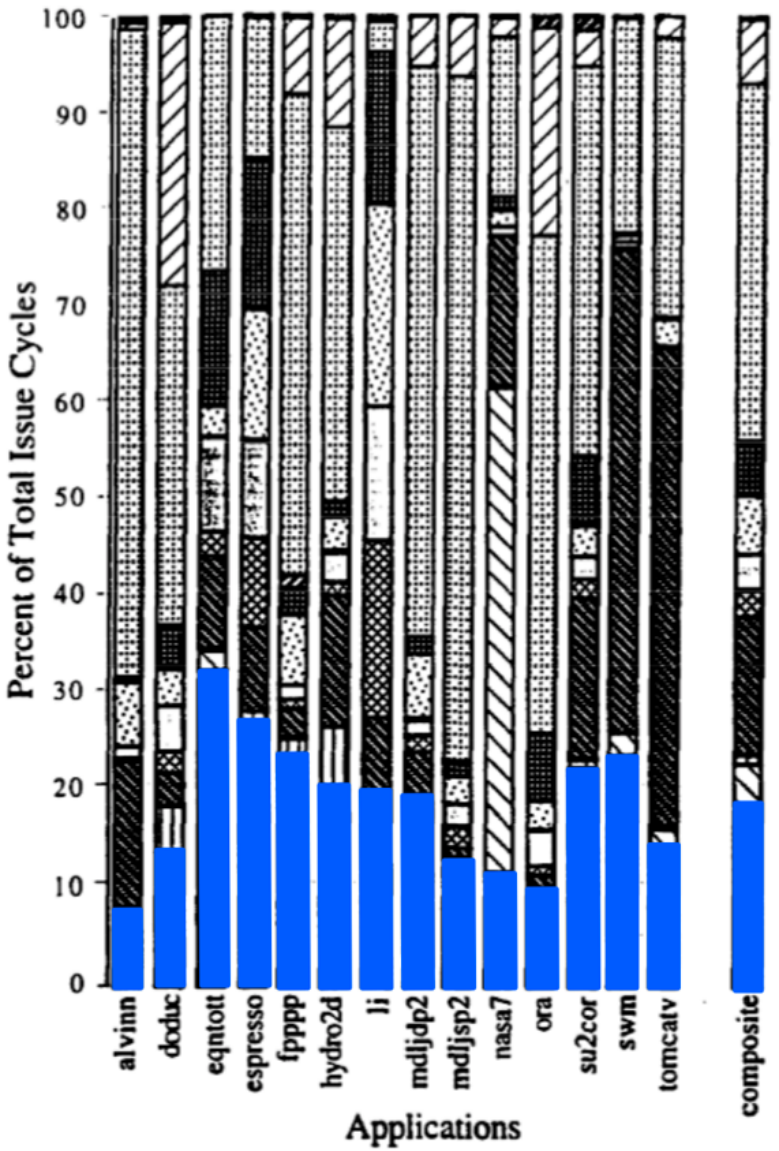
ILP Impacts

Issue Waste



- **Contributing factors**
  - **—instruction dependencies**
  - **—long-latency operations within a thread**

Simulations of 8-issue Superscalar



Simultaneous multithreading: maximizing on-chip parallelism, Tullsen et. al. ISCA, 1995.

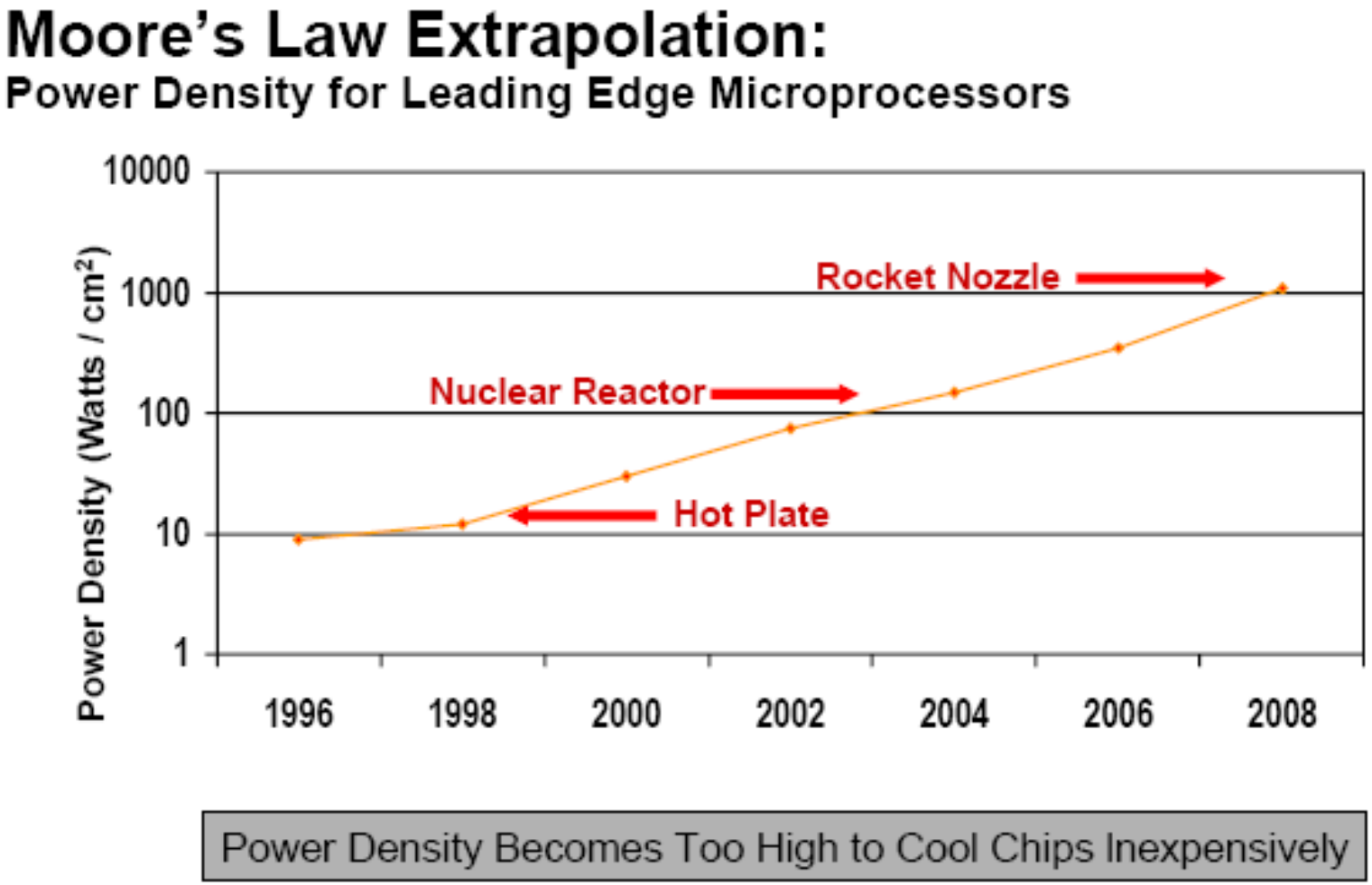
Summary:
Highly underutilized

Applications: most of SPEC92

- On average < 1.5 IPC (19%)
- Dominant waste differs by application
- Short FP dependences: 37%

Power/Heat Density Limits Frequency

- Some fundamental physical limits are being reached

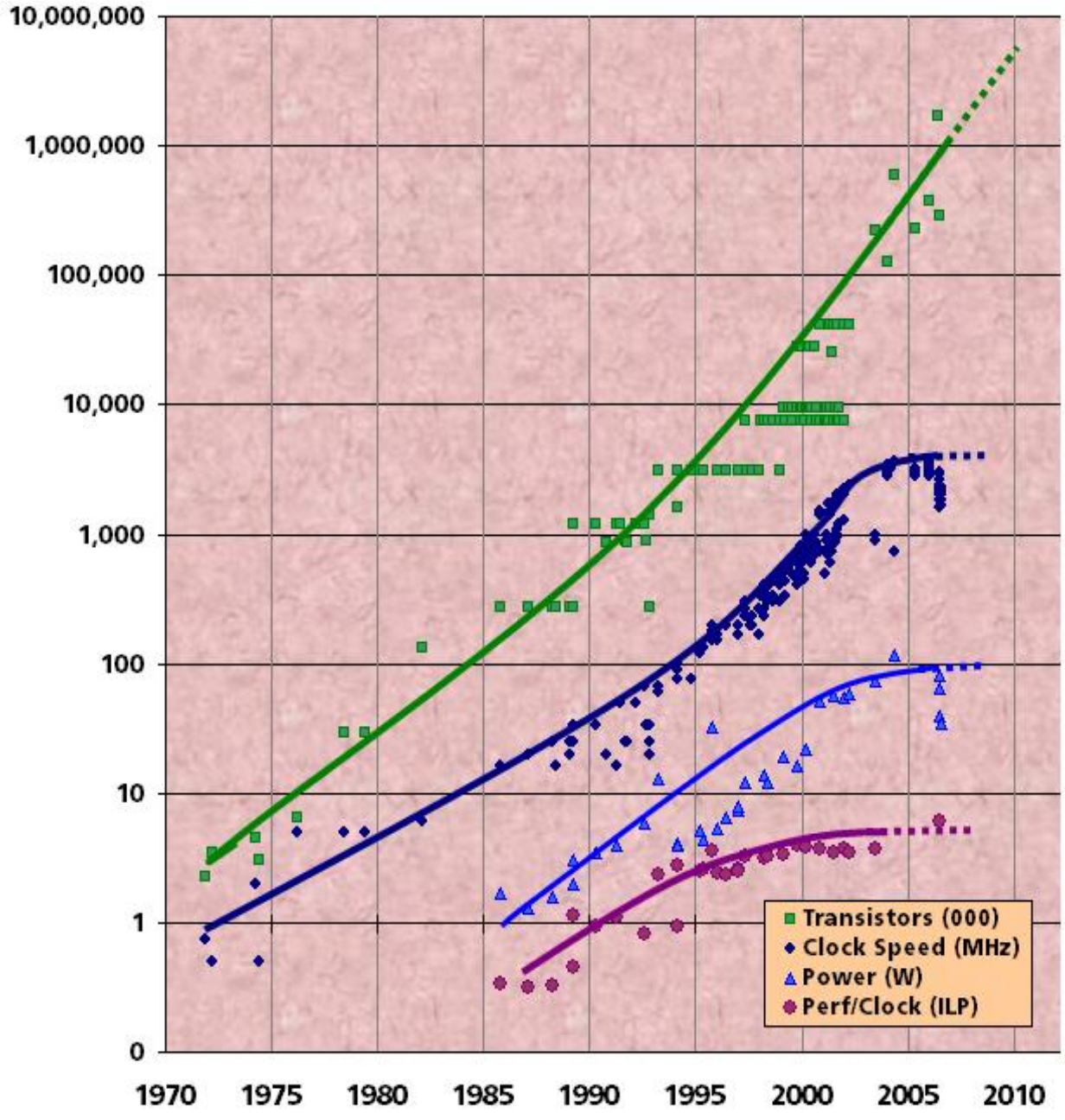


Source: Shekhar Borkar, Intel Corp

We Will Have This …

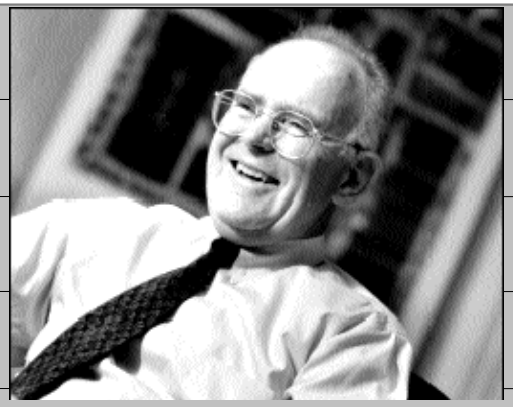
Revolution Happed Already

- Chip density is continuing increase ~2x every 2 years
  - Clock speed is not
  - Number of processor cores may double instead
- There is little or no hidden parallelism (ILP) to be found
- Parallelism must be exposed to and managed by software
  - No free lunch



Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)

The Trends



2X Transistors/Chip
Every 1.5 Years

Super Scalar/Vector/Parallel

1 PFlop/s (10^{15})
1 TFlop/s (10^{12})
1 GFlop/s (10^{9})
1 MFlop/s (10^{6})
1 KFlop/s (10^{3})

Scalar
Super Scalar
Vector
Parallel

UNIVAC 1
EDSAC 1
IBM 7090
CDC 6600
IBM 360/195
Cray 1
Cray X-MP
Cray 2
TMC CM-2
TMC CM-5
Cray T3D
ASCI Red
ASCI White Pacific
IBM BG/L

1941 1 (Floating Point operations / second, Flop/s)
1945 100
1949 1,000 (1 KiloFlop/s, KFlop/s)
1951 10,000
1961 100,000
1964 1,000,000 (1 MegaFlop/s, MFlop/s)
1968 10,000,000
1975 100,000,000
1987 1,000,000,000 (1 GigaFlop/s, GFlop/s)
1992 10,000,000,000
1993 100,000,000,000
1997 1,000,000,000,000 (1 TeraFlop/s, TFlop/s)
2000 10,000,000,000,000
2005 131,000,000,000,000 (131 Tflop/s)

1950 1960 1970 1980 1990 2000 2010

Now it's Up To Programmers

- Adding more processors doesn't help much if programmers aren't aware of them…
  - … or don't know how to use them.

- Serial programs don't benefit from this approach (in most cases).

Concluding Remarks

- The laws of physics have brought us to the doorstep of multicore technology
  - The worst or the best time to major in computer science
    - IEEE Rebooting Computing (http://rebootingcomputing.ieee.org/)
- Serial programs typically don't benefit from multiple cores.
- Automatic parallelization from serial program isn't the most efficient approach to use multicore computers.
  - Proved not a viable approach
- Learning to write parallel programs involves
  - learning how to coordinate the cores.
- **Parallel programs are usually very complex and therefore, require sound program techniques and development.**

References

- Introduction to Parallel Computing, Blaise Barney, Lawrence Livermore National Laboratory
  - https://computing.llnl.gov/tutorials/parallel_comp
- Some slides are adapted from notes of Rice University John Mellor-Crummey's class and Berkely Kathy Yelic's class.
- Examples are from chapter 01 slides of book "An Introduction to Parallel Programming" by Peter Pacheco
  - Note the copyright notice
- Latest HPC news
  - http://www.hpcwire.com
- World-wide premier conference for supercomputing
  - http://www.supercomputing.org/, the week before thanksgiving week

Vision and Wisdom by Experts

- "I think there is a world market for maybe five computers."
  - Thomas Watson, chairman of IBM, 1943.
- "There is no reason for any individual to have a computer in their home"
  - Ken Olson, president and founder of Digital Equipment Corporation, 1977.
- "640K [of memory] ought to be enough for anybody."
  - Bill Gates, chairman of Microsoft,1981.
- "On several recent occasions, I have been asked whether parallel computing will soon be relegated to the trash heap reserved for promising technologies that never quite make it."
  - Ken Kennedy, CRPC Directory, 1994

Linus: The Whole "Parallel Computing Is The Future" Is A Bunch Of Crock.

WEDNESDAY, DECEMBER 31, 2014 AT 9:09AM

http://highscalability.com/blog/2014/12/31/linus-the-whole-parallel-computing-is-the-future-is-a-bunch.html

A simple example

- Compute n values and add them together.
- Serial solution:

```
sum = 0;
for (i = 0; i < n; i++) {
   x = Compute_next_value(. . .);
   sum += x;
}
```

Example (cont.)

- We have p cores, p much smaller than n.
- Each core performs a partial sum of approximately n/p values.

```
my_sum = 0;
my_first_i = . . . ;
my_last_i = . . . ;
for (my_i = my_first_i; my_i < my_last_i; my_i++) {
   my_x = Compute_next_value( . . .);
   my_sum += my_x;
}
```

Each core uses it's own private variables and executes this block of code independently of the other cores.

Example (cont.)

- After each core completes execution of the code, is a private variable my_sum contains the sum of the values computed by its calls to Compute_next_value.

- Ex., 8 cores, n = 24, then the calls to Compute_next_value return:

1,4,3, 9,2,8, 5,1,1, 5,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9

Example (cont.)

- Once all the cores are done computing their private my_sum, they form a global sum by sending results to a designated “master” core which adds the final result.

Example (cont.)

```
if (I'm the master core) {
    sum = my_x;
    for each core other than myself {
        receive value from core;
        sum += value;
    }
} else {
    send my_x to the master;
}
```

SPMD: All run the same program, but perform differently depending on who they are.

Example (cont.)

Core	0	1	2	3	4	5	6	7
my_sum	8	19	7	15	7	13	12	14

Global sum

8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95

Core	0	1	2	3	4	5	6	7
my_sum	95	19	7	15	7	13	12	14

But wait!

There's a much better way to compute the global sum.

Better parallel algorithm

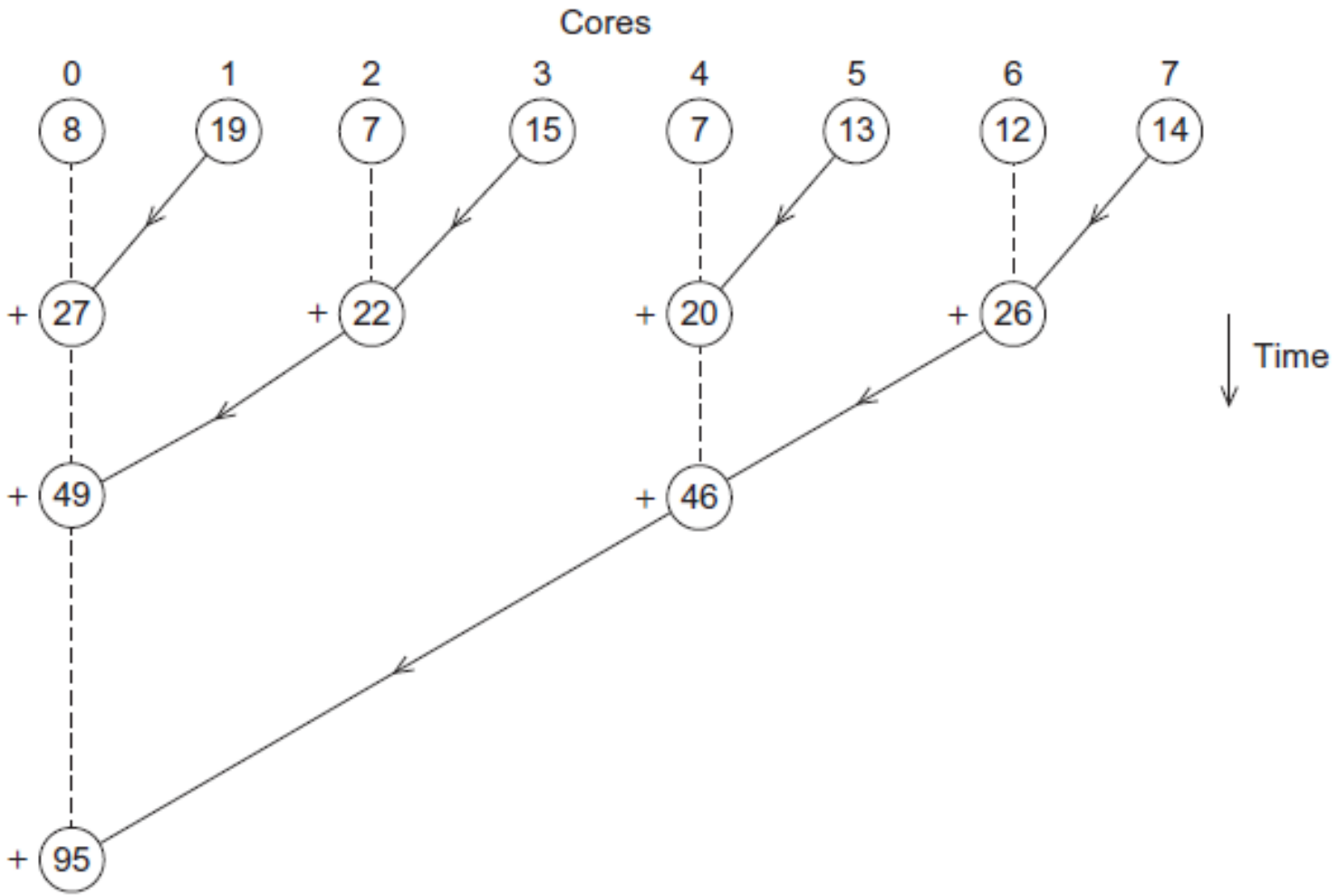
- Don't make the master core do all the work.
- Share it among the other cores.
- Pair the cores so that core 0 adds its result with core 1's result.
- Core 2 adds its result with core 3's result, etc.
- Work with odd and even numbered pairs of cores.

Better parallel algorithm (cont.)

- Repeat the process now with only the evenly ranked cores.
- Core 0 adds result from core 2.
- Core 4 adds the result from core 6, etc.

- Now cores divisible by 4 repeat the process, and so forth, until core 0 has the final result.

Multiple cores forming a global sum



Analysis

- In the first example, the master core performs 7 receives and 7 additions.
- In the second example, the master core performs 3 receives and 3 additions.
- The improvement is more than a factor of 2!

Analysis (cont.)

- The difference is more dramatic with a larger number of cores.
- If we have 1000 cores:
  - The first example would require the master to perform 999 receives and 999 additions.
  - The second example would only require 10 receives and 10 additions.

- That's an improvement of almost a factor of 100!