Introduction au calcul haute performance

Master GENIAL (Génie Informatique en Alternance), Université de Paris

Pierre Kestener

CEA, DSSI/SANL

Université de Paris, Grands Moulins, Mai 2022

Déroulement du cours d'introduction au HPC

- **cours+TP** (13/05/22): Introduction au HPC + MPI
- **cours+TP** (20/05/22): Programmation en mémoire partagée + programmation GPU
- **cours+TP** (03/06/22): programmation GPU suite + modèles de programmation unifiés CPU/GPU (OpenACC / Kokkos + python)

- **Les ressources de calcul utilisées pendant les TP:**
  - poste de travail
  - machine `odette` (Univ. Paris, UFR Info)
- **Evaluation du cours**
- **Horaire du cours : 9h00 - 17h30**

Déroulement du cours - Introduction MPI

- **Où trouver les planches et autre matériel du cours ?**

  https://gaufre.informatique.univ-paris-diderot.fr/kestener/m2-genial-hpc

- **On y trouve quoi ?**
  - Un document *aide-mémoire* sur l'interface de programmation MPI (C et Fortran): `mpi_aide_memoire_F90.pdf` et `mpi_aide_memoire_C.pdf`
  - les planches sur HPC, MPI, CUDA, OpenMP/OpenACC dans `slides`
  - des exemples de code en C et fortran dans `mpi/code`
  - de la documentation supplémentaire sur C/fortran dans `doc`

Recommended reading on MPI / Parallel Programming

- MPI standard:

  http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report/mpi31-report.htm (2015)

  https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf (2021)
- books:
  - **Parallel Programming - for multicore and cluster systems** by T. Rauber and G. Rünger, Springer, 2010
  - **Multicore Application Programming** - For Windows, Linux and Oracle Solaris by Darryl Gove, Addison-Wesley, 2010
  - From Texas Advanced computing center, Victor Eijkhout's mpi course and mpi-3 course
- on-line:
  - Cours MPI de l'IDRIS
  - https://mpitutorial.com/tutorials/
  - Parallel computing tutorial at LLNL

Recommended reading on Linux environment

- **Aide mémoire Linux / bash :**
  - Learn/improve your skill on Linux's command line/Bash e.g. http://swcarpentry.github.io/shell-novice/
  - http://www.tldp.org/LDP/abs/html/
  - http://www.epons.org/commandes-base-linux.php
- **Améliorer ses compétences:**
  Linux-101-Hacks.pdf, by Ramesh Natarajan
- Ne JAMAIS hésiter à lire la page de manuel en ligne d'une commande ou d'une API inconnue, e.g. :
  `man MPI_Send`

Recommended reading on HPC in general

- Books: Parallel Programming for Science and Engineering, by Victor Eijkhout
- Good idea to check / read on a regular basis websites like:
  - subscribe blog/news letters on HPC; e.g. `https://www.nextplatform.com/`, `http://www.admin-magazine.com/HPC/Articles`
- Hardware vendors blogs:
  - Intel Parallel Universe Magazine `http://software.intel.com/en-us/intel-parallel-universe-magazine` (attention, parfois un peu biaisé, à lire avec discernement)
  - IXPUG (Intel Extreme Performance User Group)
  - Nvidia's developer blog

Other general recommendations ...

... to a software developper for scientific applications

- Learn/improve your skill on Linux's command line/Bash
  e.g. http://swcarpentry.github.io/shell-novice/,
  http://www.tldp.org/LDP/abs/html/,
  http://www.epons.org/commandes-base-linux.php
- Learn GIT !
  - Get yourself a github or Bitbucket account
  - Read/Watch a tutorial to get started, e.g.
    https://www.grafikart.fr/formations/git
- Learn a text editor like vim or emacs (at advanced level, i.e. learn keyboard short-cuts), or maybe vscode
- Learn python (data analysis, visualization tools, ...)
- Learn C++11/17, follow recent evolution to introduce parallelism at core language
- Have some good knowledge of hardware and the linux operating system
- Never stop learning...
- Use multiple sources, multiple points of view...

A few links about python for scientific computing ...

- Learn to use a linux distro independant python, e.g. miniconda + use conda environments
- Learn to use Jupyter Lab for running python notebooks inside a web browser
- https://github.com/barbagroup/CFDPython
- https://github.com/ipython-books/cookbook-2nd-code
- Two very good books about IPython by Cyrille Rossant:
  - http://cyrille.rossant.net/books
- http://sbu-python-class.github.io/python-science/
- https://github.com/ContinuumIO/gtc2020-numba : python+cuda tutorial

A few links about C++

- A recently published book: `Discovering Modern C++: An Intensive Course for Scientists, Engineers, and Programmers`, and companion github website
- Some presentations made at CppCon (C++ annual conference): state of the art of C++ language
- List of Lists of C++ related resources:
  - `https://github.com/fffaraz/awesome-cpp`
  - `https://github.com/fffaraz/awesome-cpp/blob/master/books.md`
- What should I do (among others things) to improve my C++ skills ? Read code written by others is very good.
- Google C++ style guide, a must read

A few links about C++ - Advanced

- Books by Herb Sutter: C++ coding standards, Exceptional C++, More exceptional C++; these books are quite old (<= 2004) but still of very good advise to read.
- Book by Scott Meyers: Effective Modern C++, for those would already know well C++, but want insightfull tips to use efficiently new C++11 features (lvalue/rvalue, std::move, auto, lambda, std::thread, ...)
- C++ Idioms list: e.g. What is Pimpl ?
- Design patterns; see also book by Gamma et al.
  Hands-On Design Patterns with C++, by F. G. Pikus, Packt Publisher
  Most used patterns: factory, abstract factory, visitor, ...

Other links about computer science in general

- Learn Modern CMake
  CMake Cookbook by R. Bast and R. Di Remigio, Packt publisher, and companion website
- https://github.com/vhf/free-programming-books/blob/master/free-programming-books-fr.m
- https://github.com/Michael0x2a/curated-programming-resources/blob/master/resources.md
- https://github.com/jnv/lists

Other recommendations ... computationnal physics

- **Watch videos made by W. Bangerth** (main author of Deal.II): `https://www.math.colostate.edu/~bangerth/videos.html`
- **Deal.ii** is a very good reference about implementing high-order Finite Element Methods, Discontinuous Galerkin Methods in a HPC context (MPI distributed). Deal.ii has some specific features not necessarily found in other competitive frameworks: hexahedral meshes, adaptive mesh refinement (by leveraging library p4est).
- Do not hesitate to test the tutorial examples
- Overview of parallelization inside deal.ii
- How to build Deal.ii on a desktop linux machine ?

Misc ou pas

- Cloud Computing / Big Data /IoT
- Docker: une solution de machine virtuelle légère / container
  - une video de présentation de Docker orientée Cloud: https://www.youtube.com/watch?v=XgKOC6X8W28
  - Quel intérêt pour le HPC ? http://openpowerfoundation.org/blogs/using-docker-in-high-perf
    http://www.admin-magazine.com/HPC/Articles/Singularity-A-Conta
- Envie d'essayer ?
  - Installer Docker sous linux, visualiser un tutorial (e.g. celui de Grafikart.fr)
  - Exemple de script de création d'un conteneur Docker pour Trilinos: https://hub.docker.com/r/johntfoster/trilinos/~/dockerfile/
  - autre exemple: https://trilinos.org/packages/web-trilinos/webtrilinos-docker-

- **Objectifs de ce cours:**
  - Pas juste une introduction à MPI, OpenMP, CUDA, Kokkos, SyCL
  - Qu'est ce que le HPC (High Performance Computing) ?
  - Qu'est ce qu'un supercalculateur ? Spécificités Hardware / Software ?
  - *Parallel Computing*
  - Des notions de bases sur le matériel: multi-cœurs, multi-thread, mémoire cache ...
  - Les modèles de programmation parallèle: MPI, OpenMP, multi-thread, ...
  - **Exercices pratiques**
  - Outils d'analyse de performance et d'aide à la parallélisation
  - Certains sujets seront survolés, mais des pointeurs externes pour approfondir seront fournis

Introduction

- Parallel computing : definition(s)
- Concurrence / parallélisme;
- les architectures matérielles de calcul; tendances actuelles
- Loi d'Amdhal; notions de Weak/Strong scaling
- Qu'est ce qu'un séquenceur de travaux ?

Parallel Computing: definition(s)

Parallel computing: using multiple processors in parallel to solve problems more quickly than with a single processor

Figure: source: M. Zahran, NYU

Parallel Computing: definition(s)

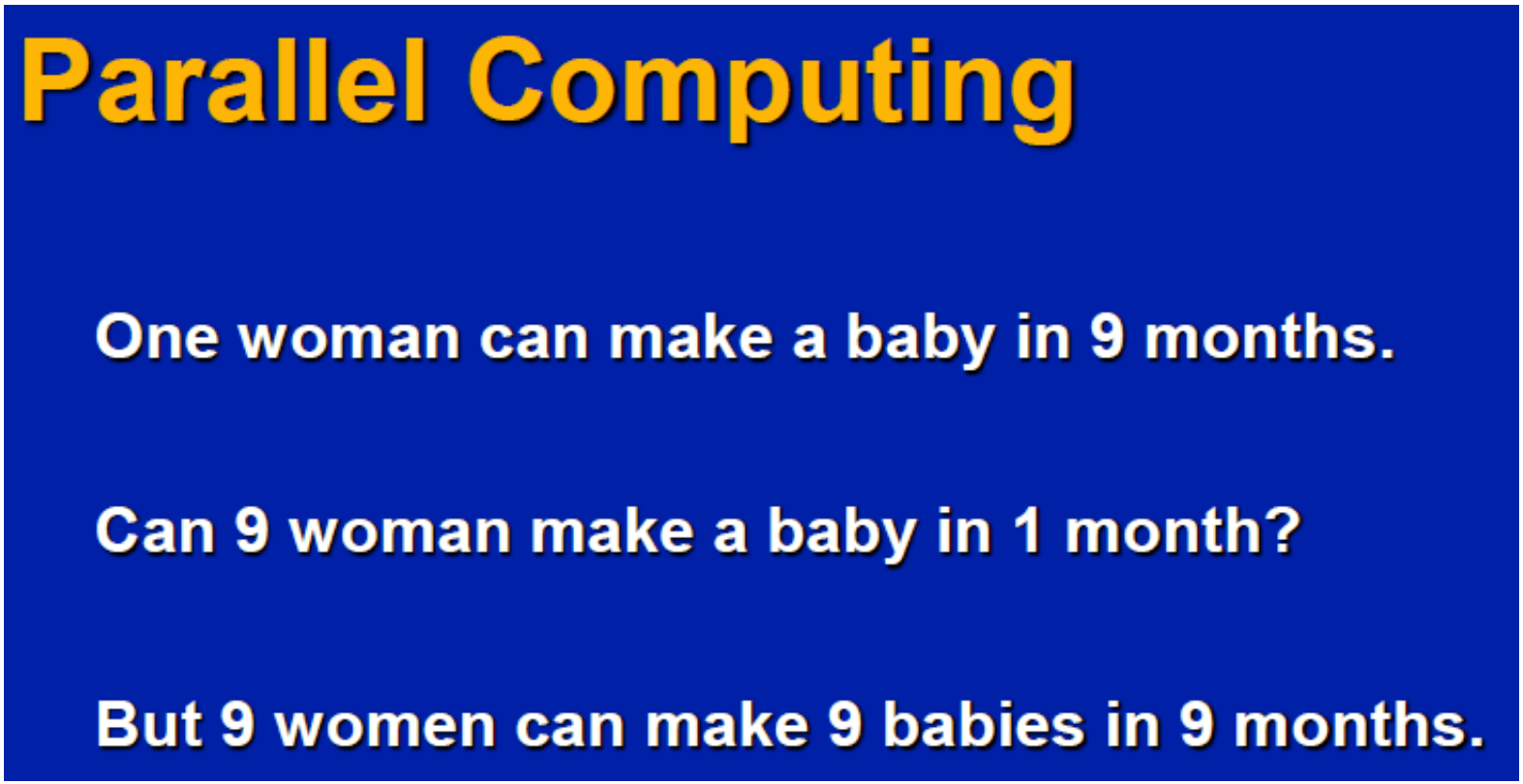
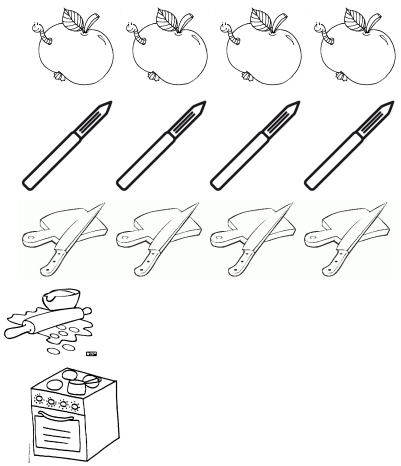


Figure: source: John Urbanic, Pittsburgh Supercomputing Center

Devinette #1

Combien de temps pour faire une tarte aux pommes



$4 \times$ 1 minutes

$4 \times$ 1 minutes

1×5 minutes

1×30 minutes

---

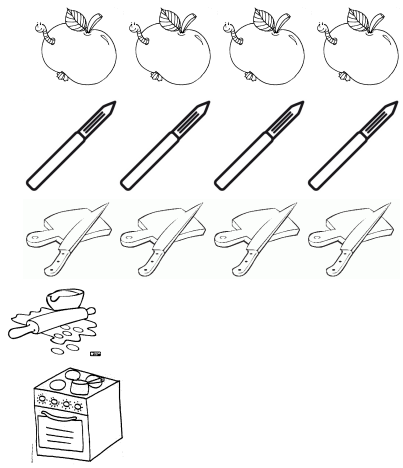
43 minutes tout seul

---

xx minutes à 2 ?

Devinette #2

Combien de temps pour faire une tarte aux pommes



4 × 1 minutes

4 × 1 minutes

1 × 5 minutes

1 × 30 minutes

---

37 minutes à 2

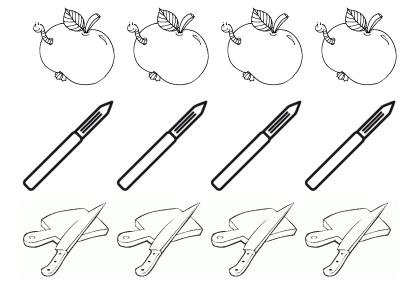
---

xx minutes à 4 ?

http://serge.liyun.free.fr/serge/sources/cours_parallelism.pdf

Devinette #3

Combien de temps pour faire une tarte aux pommes



4 × 1 minutes

4 × 1 minutes

1 × 5 minutes

1 × 30 minutes

---

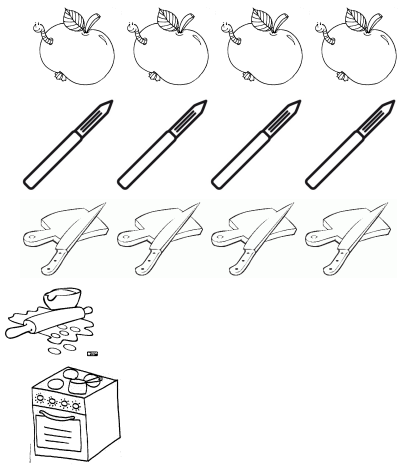
35 minutes à 4

---

xx minutes à 3 avec un seul couteau et un seul économe ?

Devinette #4

Combien de temps pour faire une tarte aux pommes



$4 \times$ 1 minutes

$4 \times$ 1 minutes

1×5 minutes

1×30 minutes

---

35 minutes à 3

---

et si peu de temps pour tout manger. . .

Calcul Parallèle : définition(s)

Une tarte aux pommes, c'est quoi ?
Un assemblage de 3 choses:

- **des ingrédients:** pommes, sucre, farine, eau, sel, cannelle, ...
- **une recette:** qu'est qu'on doit faire et dans quel ordre ? Où sont les pommes ? Où est l'épluchEur ? Penser à faire préchauffer le four, ...
- **un pâtissier + une cuisine**: celui qui travaille, qui exécute la recette !

Calcul Parallèle : définition(s)

Un programme informatique, c'est quoi ?
Un assemblage de 3 choses:

- ~~**des ingrédients**~~ **des données:** des fichiers, des données envoyées sur le réseau, ...
- ~~**une recette**~~ **un algorithme:** qu'est qu'on doit faire et dans quel ordre ? Où sont les données ? Quel est le nom du fichier ?
- ~~**un pâtissier + une cuisine**~~
  un programme + un processeur: celui qui travaille, qui exécute le programme !

Calcul Parallèle : définition(s)

Parallel computing: using multiple processors in parallel to solve problems more quickly than with a single processor

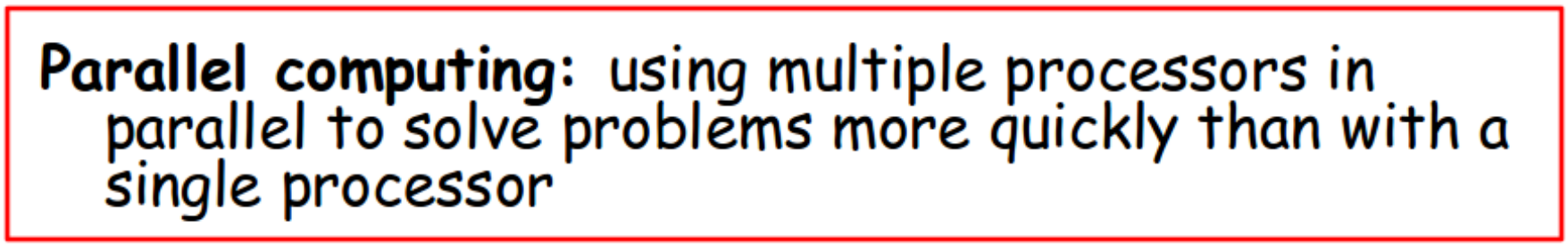


Figure: source: M. Zahran, NYU

traduction possible: Utiliser plusieurs patissiers pour faire une tarte aux pommes plus rapidement...

Calcul Parallèle : définition(s)

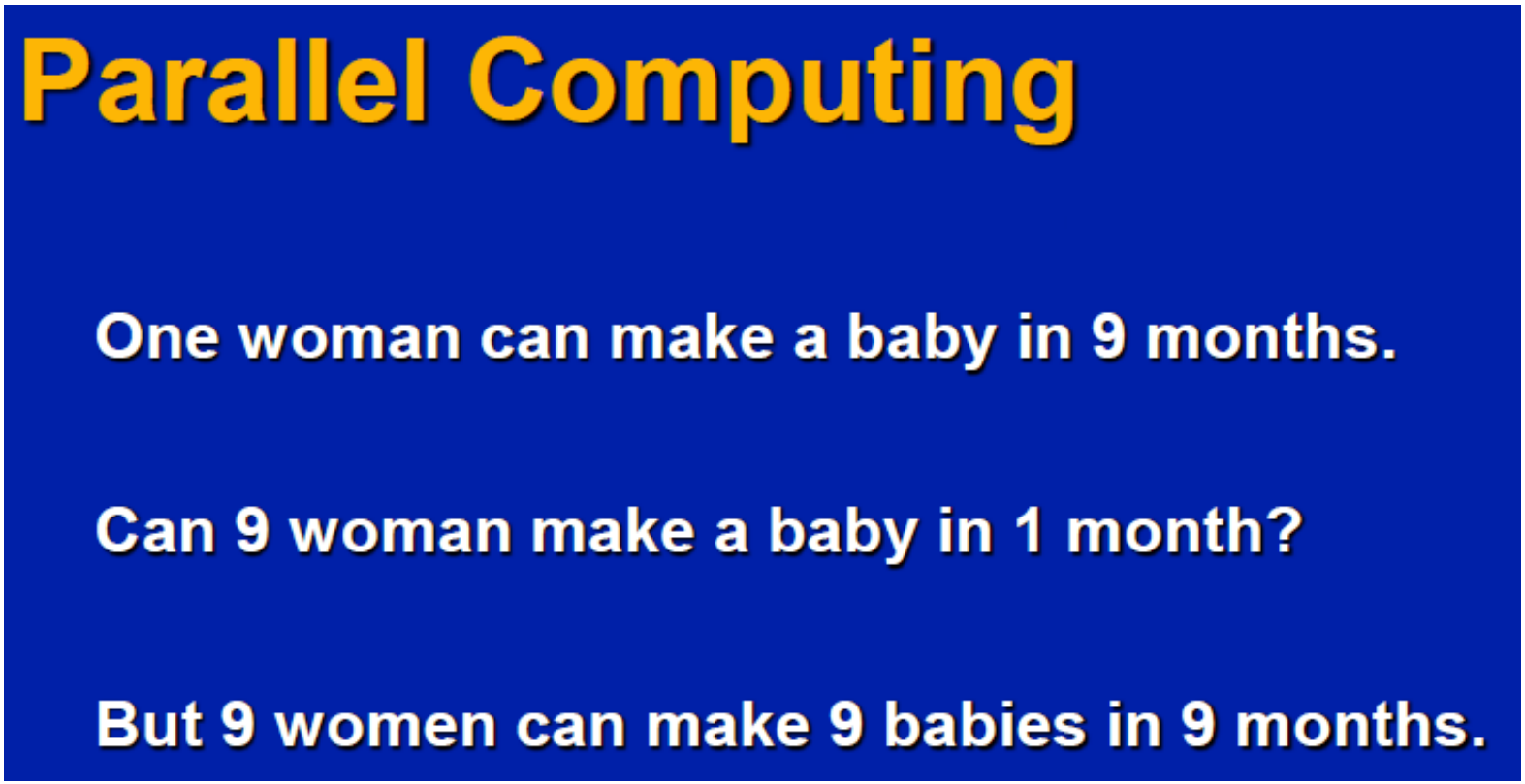


Figure: source: John Urbanic, Pittsburgh Supercomputing Center

traduction possible: faire un bébé, c'est déjà compliqué, alors plusieurs...

Parallel Computing: definition(s)

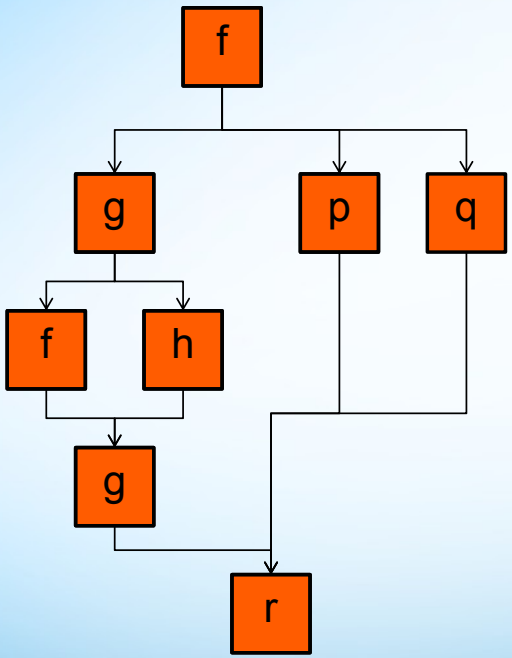
Types de parallélisme

- Task parallelism
- Data parallelism
- Pipeline parallelism

  http://www.futurechips.org/parallel-programming-2/parallel-programming-clarifying-pi
- exemple : correction de copies d'examen

De la recette de cuisine à la notion de concurrence

Superscalar Sequence



Developer writes "serial" code:

```
B = f(A);
C = g(B);
E = f(C);
F = h(C);
G = g(E,F);
P = p(B);
Q = q(B);
R = r(G,P,Q);
```

- However, tasks only need to be ordered by data dependencies
- Depends on limiting scope of data dependencies

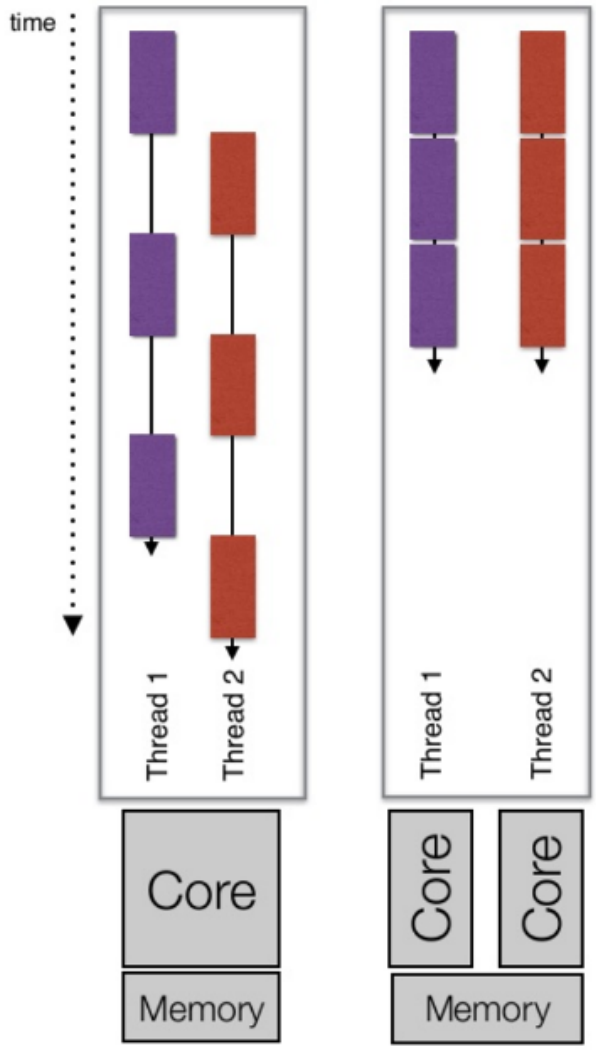
Software & Services Group, Developer Products Division

7

ref: https://www.usenix.org/legacy/event/hotpar10/tech/slides/mccool.pdf

Parallelism ≠ Concurrency

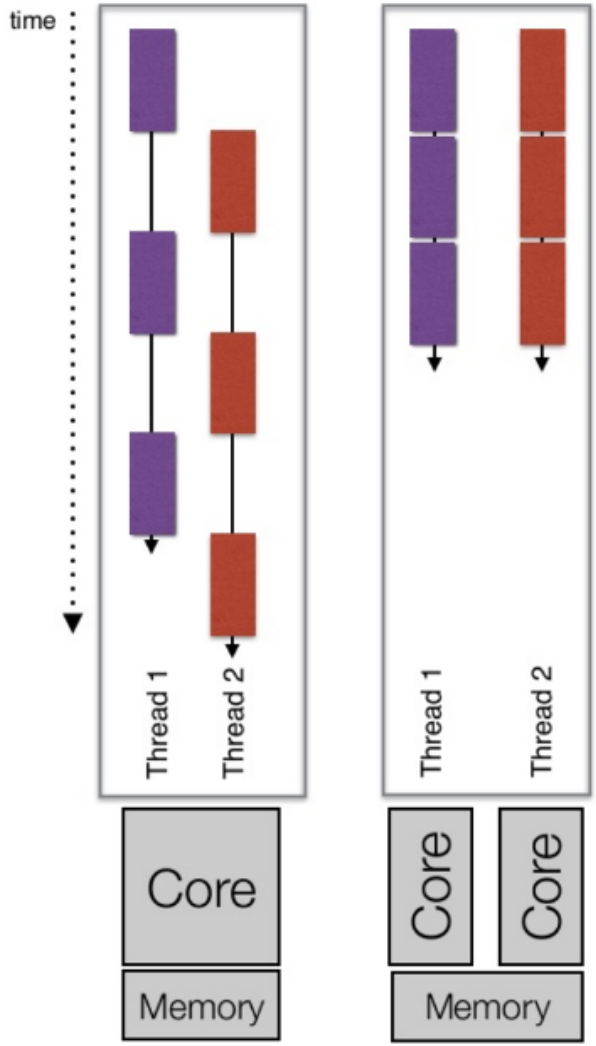
- **Concurrency:** At least two tasks are making progress at the same time frame.
  - Not necessarily at the same time
  - Include techniques like time-slicing
  - Can be implemented on a single processing unit
  - Concept more general than parallelism
  - Ex: multi-tasking on a single-core (time multiplexing)
- **Parallelism:** At least two tasks execute literally at the same time.
  - Requires hardware with multiple processing units
- If you program using threads (concurrent programming), it's not necessarily going to be executed as such (parallel execution), since it depends on whether the machine can handle several threads (multi-core - hardware thread).



source: M. Zahran, NYU

Parallelism ≠ Concurrency

- **Concurrency:** At least two tasks are making progress at the same time frame.
  - Not necessarily at the same time
  - Include techniques like time-slicing
  - Can be implemented on a single processing unit
  - Concept more general than parallelism
  - Ex: multi-tasking on a single-core (time multiplexing)
- **Parallelism:** At least two tasks execute literally at the same time.
  - Requires hardware with multiple processing units
- **Parallel computing** takes advantage of **concurrency** to:
  - Solver large problems under bounded time
  - Save on Wall clock time
  - Overcome memory constraints
  - Utilize non-local resources



source: M. Zahran, NYU

Pourquoi paralléliser ?

- **When to Parallelize:**
  - Just want efficient use of available hardware resources !
  - Program takes too long to execute on a single processor
  - Program requires too much memory to run on a single processor
  - Program contains multiple elements that are executed or could be executed independently of each other
- **Advantages of parallel programs:**
  - Single processor performance is not increasing. The only way to improve performance is to write parallel programs.
  - Data and operations can be distributed amongst N processors instead of 1 processor. Codes execute potentially N times more quickly.
- **Disadvantages of parallel programs:**
  - Greater program complexity: distributed data, task synchronization, ...
  - Productivity = (application perf) / (amount of development time)
  - Productivity reduced ? number of valid/checked/verified lines of codes per day ?

Code / Logiciel

serial code = $data + algorithm + hardware$
parallel code =?

- Conception d'un code parallèle
- Paralléliser les données ?
  - Modèle à mémoire distribuée, ex: MPI
  - Modèle à mémoire partagée (SMP), ex: OpenMP, pthread, ..
  - nouvelles problématiques, ex: cohérence de cache

Code / Logiciel

serial code = $data + algorithm + hardware$
parallel code =?

- Conception d'un code parallèle
- Paralléliser l'algorithme ? Lequel ?
  - e.g. problème à N-corps:
    algorithme naif N^2 $\Rightarrow$ **facile** à paralléliser
    algorithme en $N \log N$ (Barnes-Hut, FMM) $\Rightarrow$ **difficile** à paralléliser

Code / Logiciel

serial code = $data + algorithm + hardware$
parallel code =?

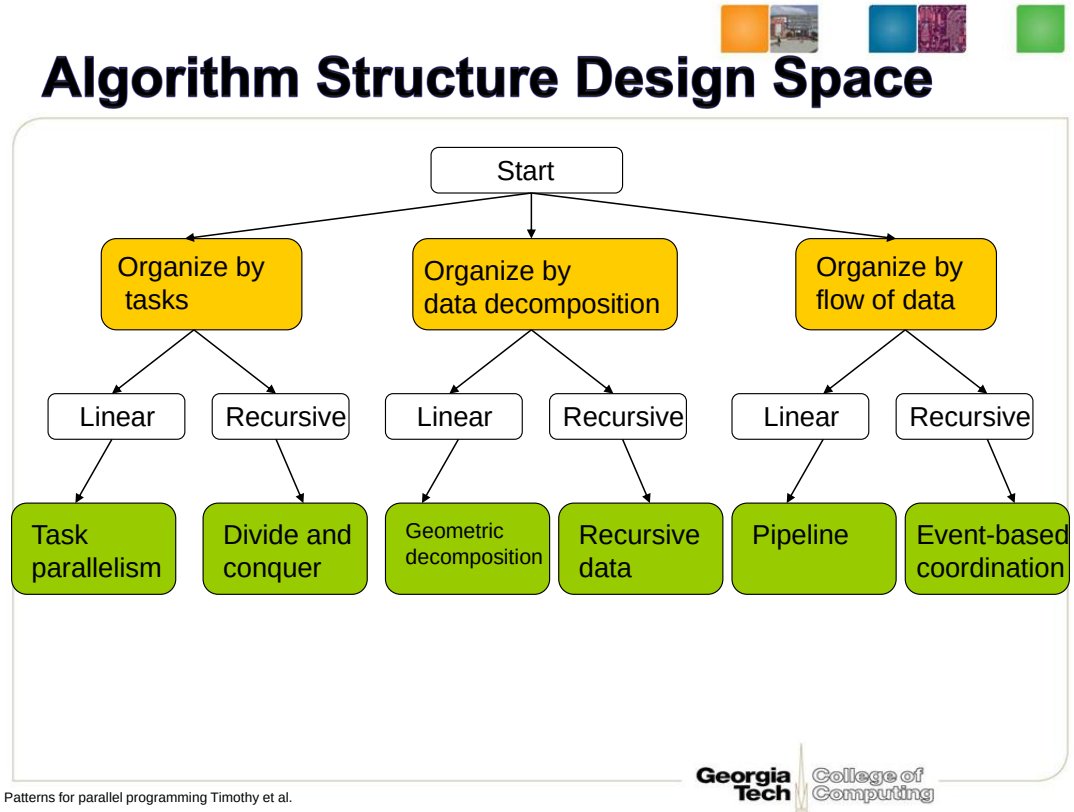
- Conception d'un code parallèle
- Hardware
  - Multiples niveaux de parallélisme et hiérarchie matérielle: cœurs *hyper-threadés*, CPU multi-cœurs, niveaux de cache L1/L2/L3, nœud multi socket, cluster
  - Utiliser un matériel spécialisé ? ou Généraliste ? Hétérogène (accélérateurs - GPU / MIC) ?
    voir l'histoire du projet GRAPE (matériel spécialisé pour le calcul des forces de type newtonien)
    `http://www.ids.ias.edu/~piet/act/comp/hardware/`

Code / Logiciel

serial code $= data + algorithm + hardware$
parallel code $=?$

- Conception d'un code parallèle
- Coûts intrinsèques de la parallélisation
  - communication / échange de données:
    - latence: temps nécessaire à démarrer une communication, indépendant de la taille des données
    - temps de transfert: après la phase de démarrage, proportionnel à la taille des données
  - compléxité des codes

Parallel programming patterns



reference: http://www.cc.gatech.edu/~hyesoon/spr11/lec_parallel_pattern.pdf

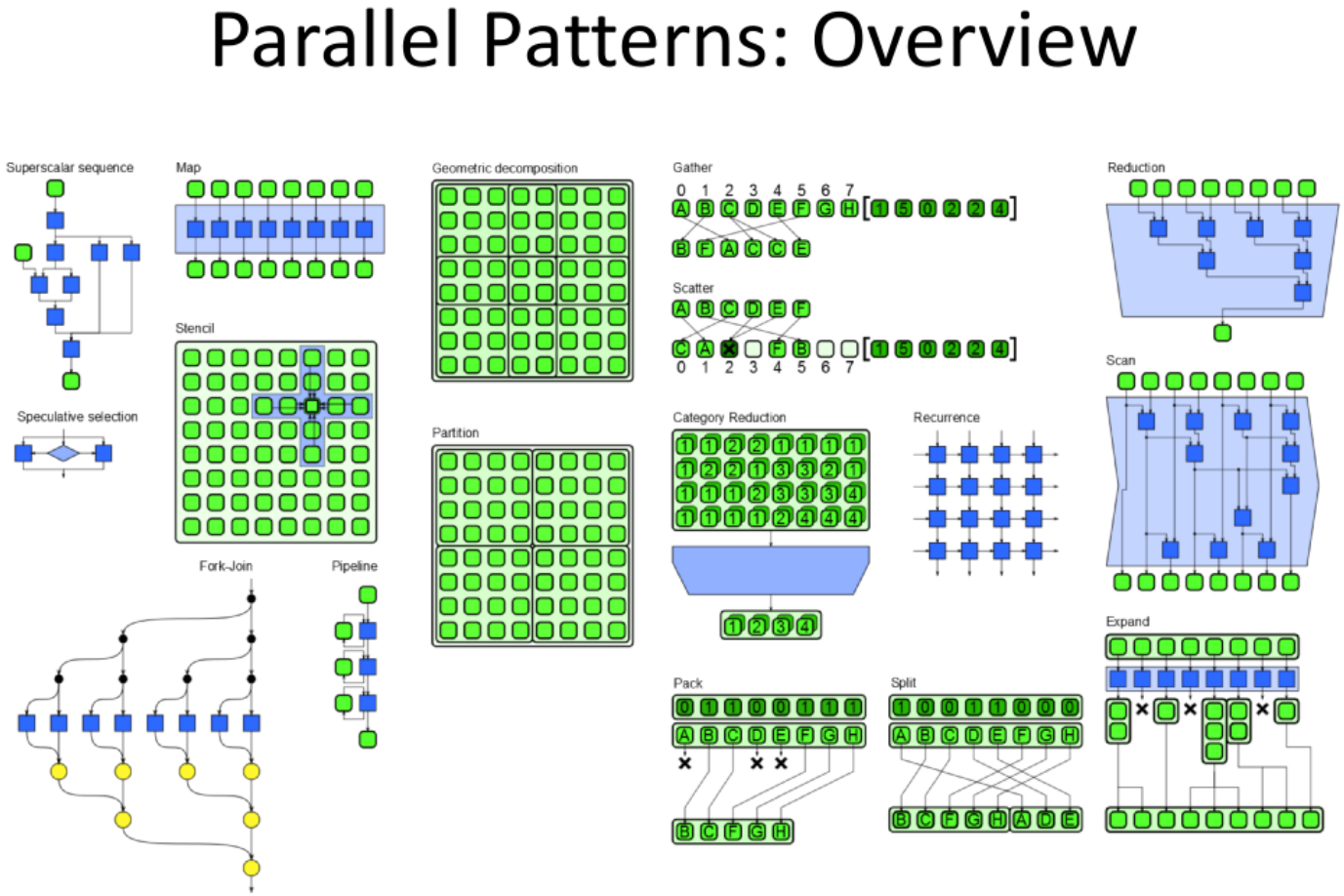
Structured Parallel Programming: Patterns for Efficient Computation by McCool, Reinders, Robinson

Parallel programming patterns

- **pattern** : **a basic structural entity of an algorithm**
- book
  Structured Parallel Programming: Patterns for Efficient Computation
- implementation: Intel TBB, and many others
- OpenMP/OpenAcc for GPU/XeonPhi: pattern-based comparison:
  map, stencil, reduce, scan, fork-join, superscalar sequence, parallel update

reference:

A Pattern-Based Comparison of OpenACC and OpenMP for Accelerator Computing



reference: Structured Parallel Programming with Patterns, SC13 tutorial, by M. Hebenstreilt, J. Reinders, A. Robison, M. McCool

(Super)computing system stack

- **Device technologies**
  - Enabling technologies for logic, memory, & communication
  - Circuit design
- **Computer architecture**
  - semantics and structures
- **Models of computation**
  - governing principles
- **Operating systems**
  - Manages resources and provides virtual machine
- **Compilers and runtime software**
  - Maps application program to system resources, mechanisms, and semantics
- **Programming**
  - languages, tools, & environments
- **Algorithms**
  - Numerical techniques
  - Means of exposing parallelism
- **Applications**
  - End user problems, often in sciences and technology

Where Does Performance Come From ?

- **Device Technology**
  - Logic switching speed and device density
  - Memory capacity and access time
  - Communications bandwidth and latency
- **Computer Architecture**
  - Instruction issue rate
    - Execution pipelining
    - Branch prediction
    - Cache management
  - Parallelism
    - Number of operations per cycle per processor : Instruction level parallelism (ILP), Vector processing
    - Number of processors per node
    - Number of nodes in a system

Emergence de la simulation dans la démarche scientifique

SciDAC (Scientific Discovery through advanced Computing)

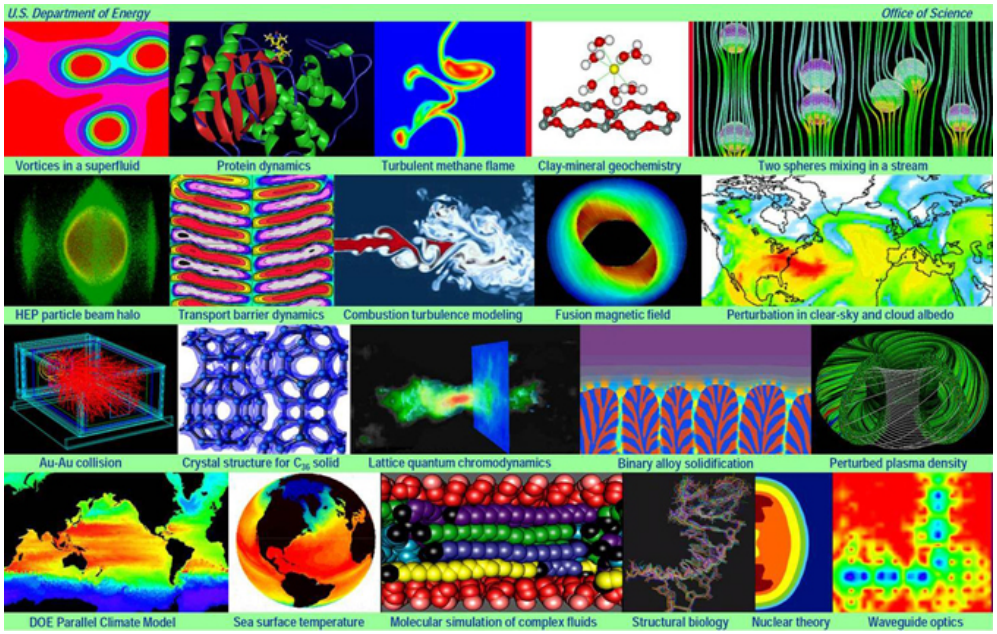


Figure: source: http://www.scidac.gov/

Emergence de la simulation dans la démarche scientifique

Pillars of science discovery:

Simulation: The Third Pillar of Science

- Traditional scientific and engineering paradigm:
  1) Do theory or paper design.
  2) Perform experiments or build system.
- Limitations:
  - 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.

2

Figure: source: Scientific Computing for engineers, CS594, J. Dongarra

Emergence de la simulation dans la démarche scientifique

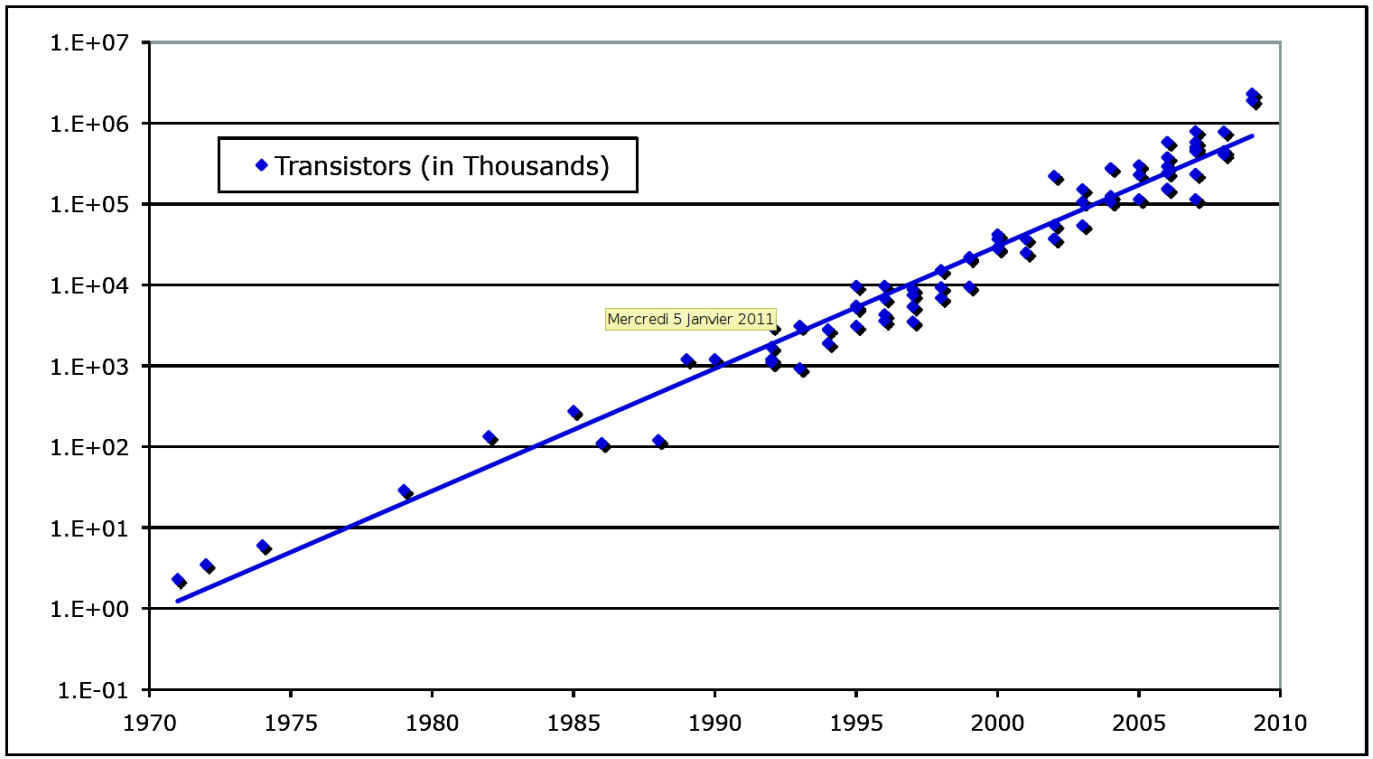
Strategic importance of supercomputing:

- essential of scientific discovery
- critical for national security
- fundamental contributor to the economy and competitiveness through use in engineering and manufacturing

source: CS594, J. Dongarra

Moore's law - *the free lunch is over...*

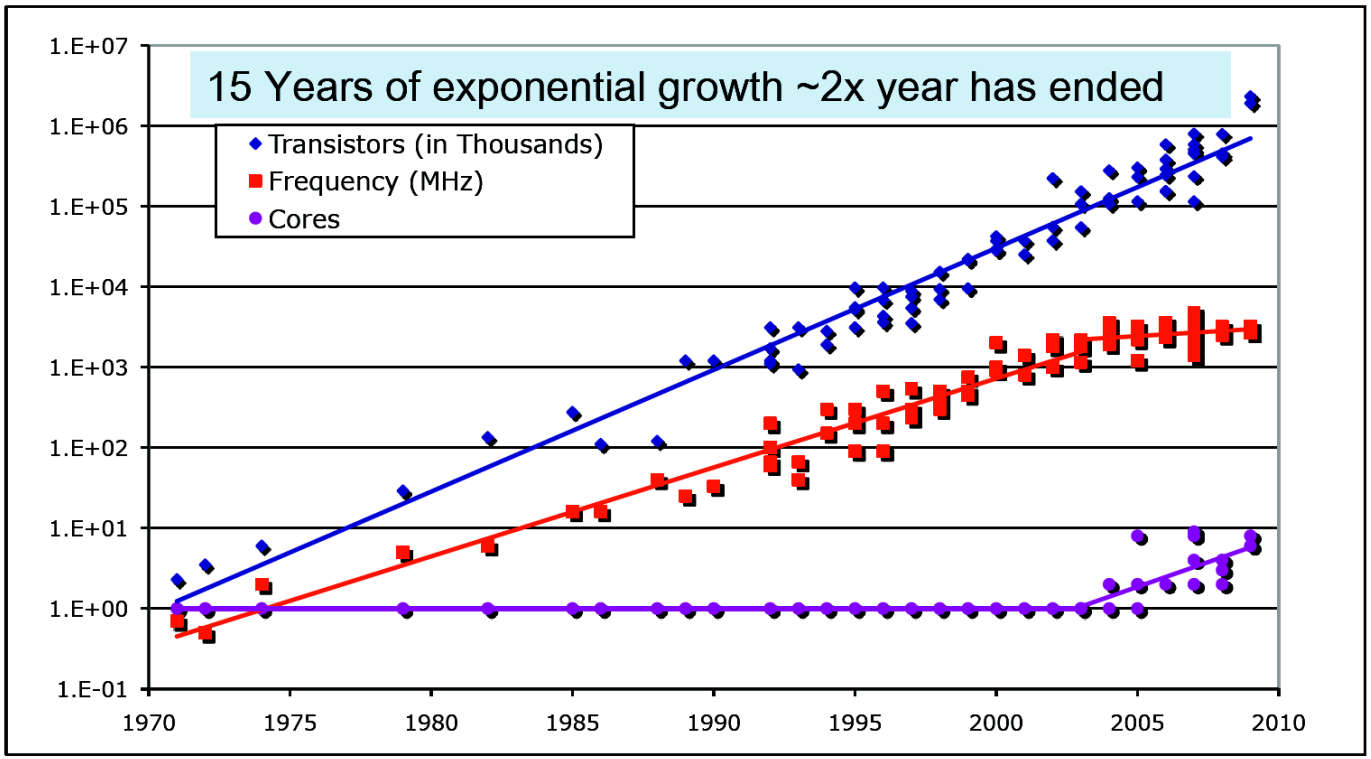
The number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years



Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç

Moore's law - *the free lunch is over...*

The number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years



Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç

Moore's law - *the free lunch is over...*

Moore's Law continues with

- **technology scaling** (32 nm in 2010, 22 nm in 2011),
- improving transistor performance to increase frequency,
- increasing transistor integration capacity to realize complex architectures,
- reducing energy consumed per logic operation to keep power dissipation within limit.

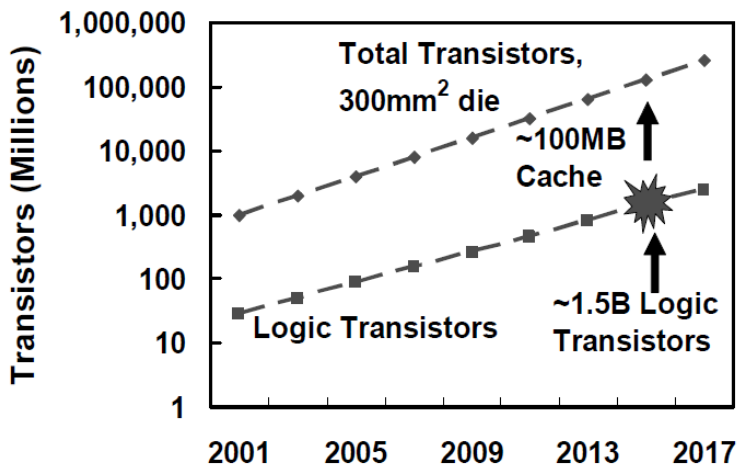


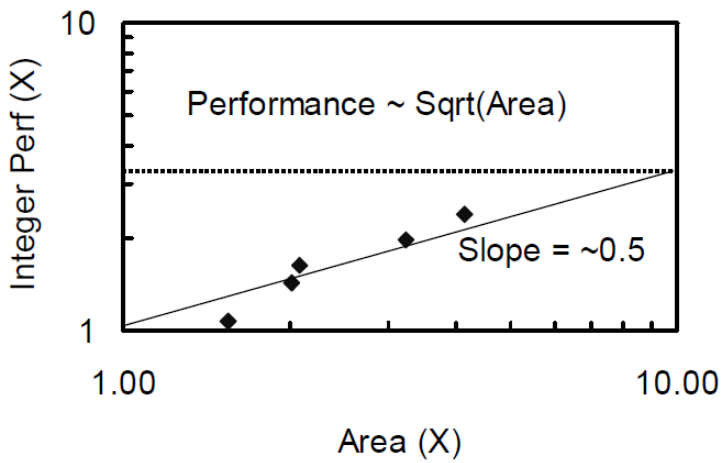
Figure 1: Transistor integration capacity

Shekhar Borkar, *Thousand Core Chips - A Technology Perspective*, in Intel Corp, Microprocessor Technology Lab, 2007, p. 1-4

Moore's law - Towards multi-core architectures

Pollack's rule - Wide adoption of multi-core architectures

- if you **double the logic** in a processor core, then it delivers **only 40% more performance**
- A multi-core microarchitecture has potential to provide near linear performance improvement with complexity and power.
- For example, **two smaller processor cores, instead of a large monolithic processor core, can potentially provide 70-80% more performance, as compared to only 40% from a large monolithic core**



Shekhar Borkar, *Thousand Core Chips - A Technology Perspective*, in Intel Corp, Microprocessor Technology Lab, 2007, p. 1-4

End of multicore scaling

Moore's law - Towards multi-core architectures

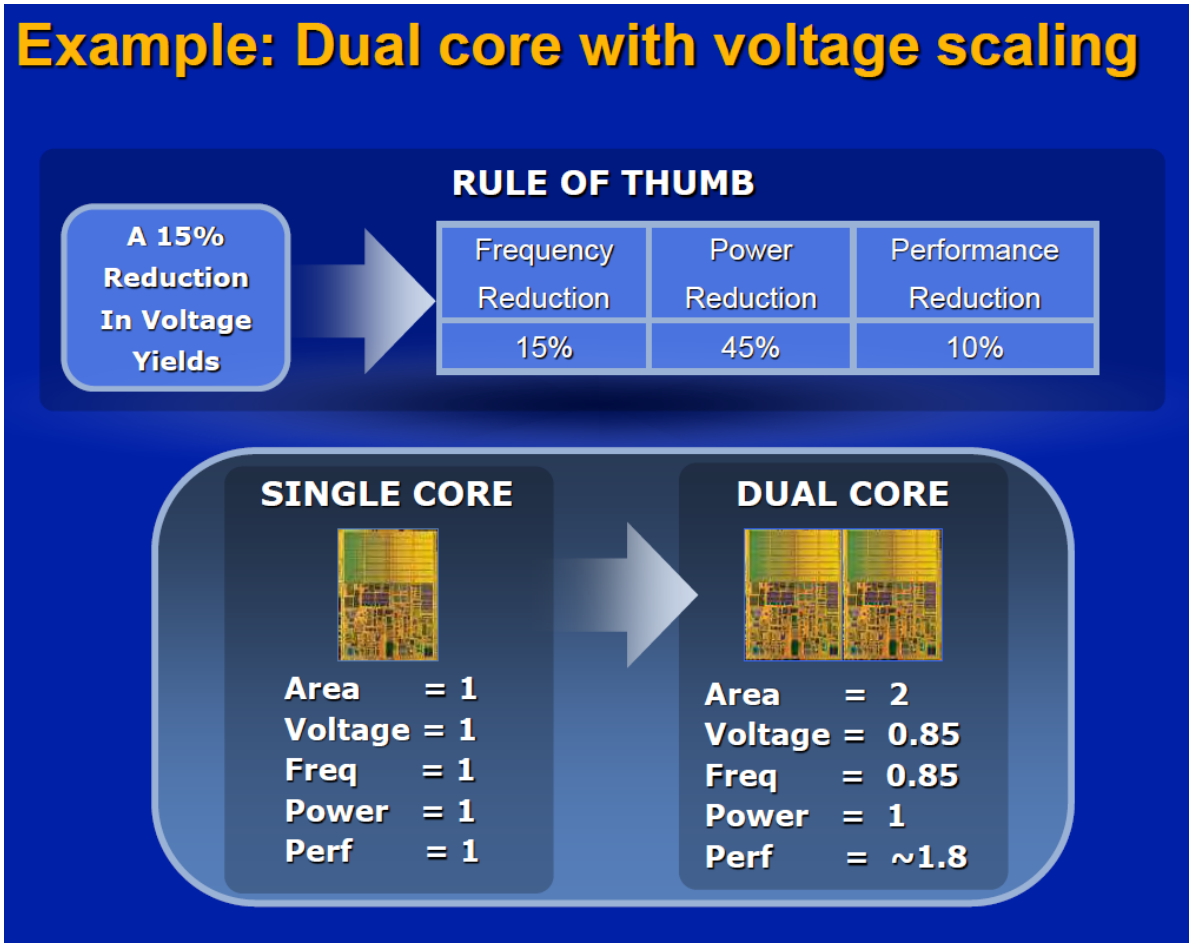


Figure: source: John Urbanic, Pittsburgh Supercomputing Center

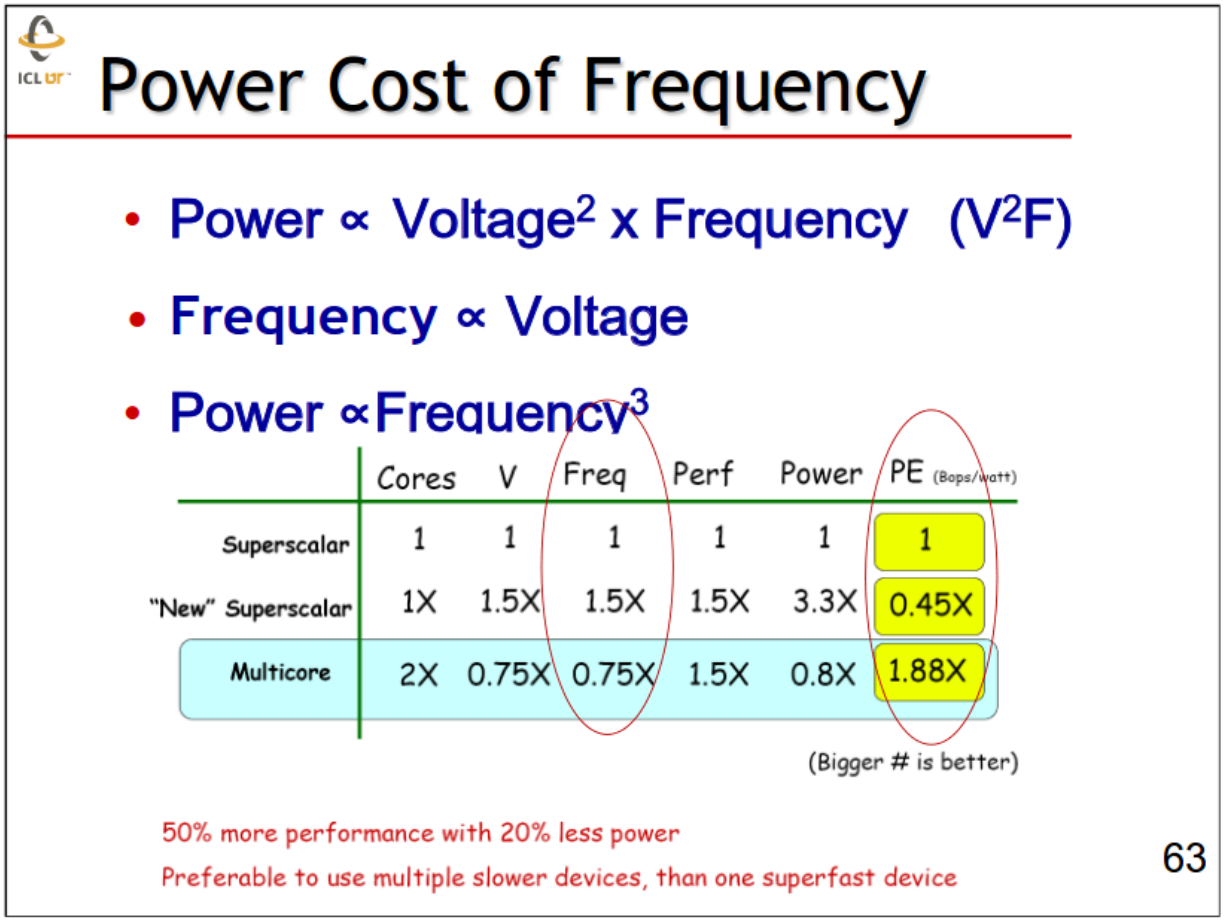


Figure: source: Scientific Computing for engineers, CS594, J. Dongarra

Moore's law - Towards multi-core architectures

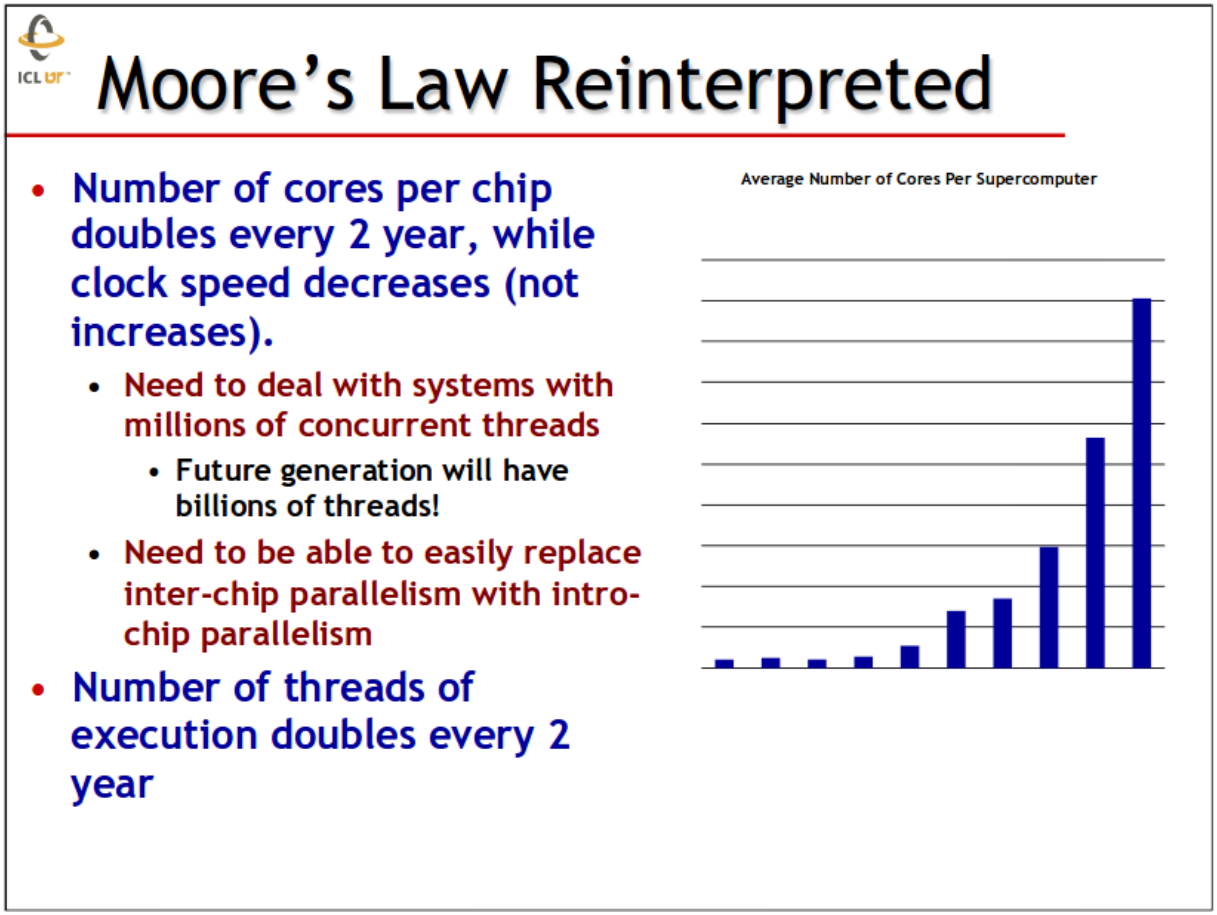


Figure: source: Scientific Computing for engineers, CS594, J. Dongarra

Moore's law - CPU/DRAM performance gap (latency)

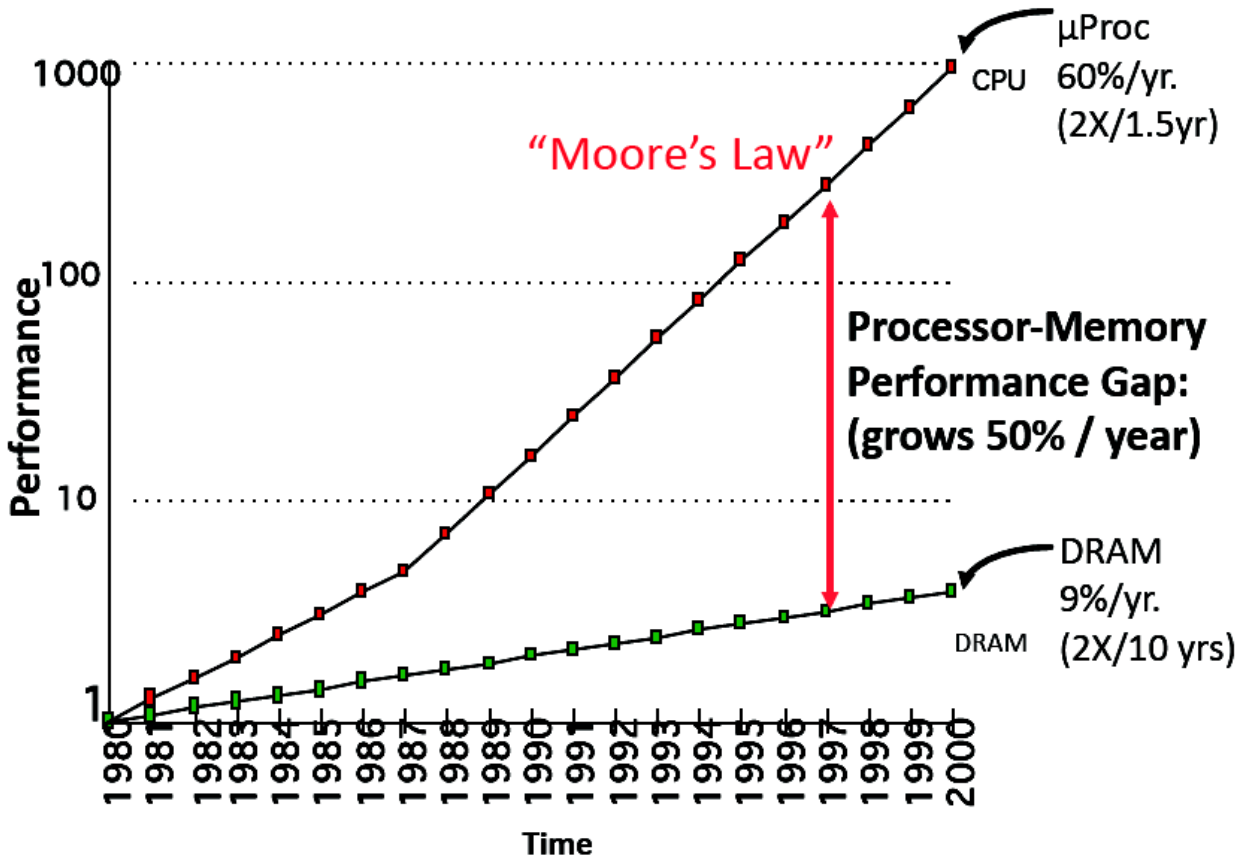


Figure: source: T. Sterling, Louisiana State University

Progrès algorithmiques

Whence new algorithms?

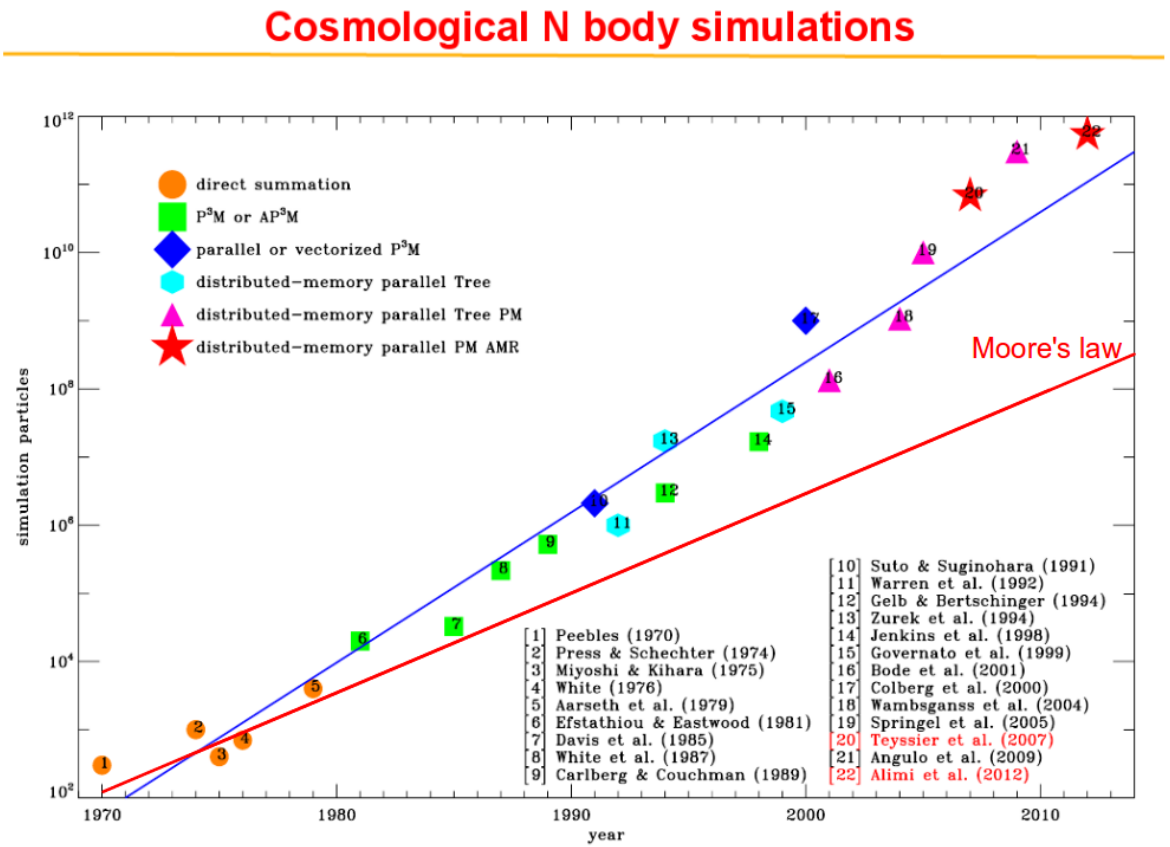
- Algorithms arise to fill the gap between *architectures that are available* and *applications that must be executed*
- Many algorithmic advances are oriented towards particular physical problems that defy the assumptions of today's optimal methods – e.g., *anisotropy, inhomogeneity, geometrical irregularity, mathematical singularity* – underlining the importance of *applied research*
- Many algorithms are *mined from the literature*, rather than *invented* –underlining the importance of *basic research*

Algorithm	Born	Why?	Reborn	Why?
Conjugate gradients	*1952*	*direct solver*	*1970s*	*iterative solver*
Schwarz Alternating procedure	*1869*	*existence proof*	*1980s*	*parallel solver*
Space-filling curves	*1890*	*topological curiosity*	*1990s*	*memory mapping function*

OSTP Briefing, 4 May 2004

source: David Keyes, prof. of applied math., Columbia

Progrès algorithmiques



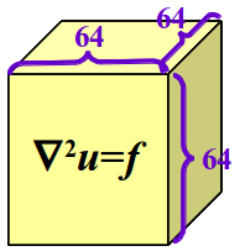
source: R. Teyssier

Progrès algorithmiques

The power of optimal algorithms

- Advances in algorithmic efficiency rival advances in hardware architecture
- Consider Poisson's equation on a cube of size $N=n^3$

Year	*Method*	*Reference*	*Storage*	*Flops*
1947	GE (banded)	Von Neumann & Goldstine	n^5	n^7
1950	Optimal SOR	Young	n^3	$n^4 \log n$
1971	CG	Reid	n^3	$n^{3.5} \log n$
1984	Full MG	Brandt	n^3	n^3



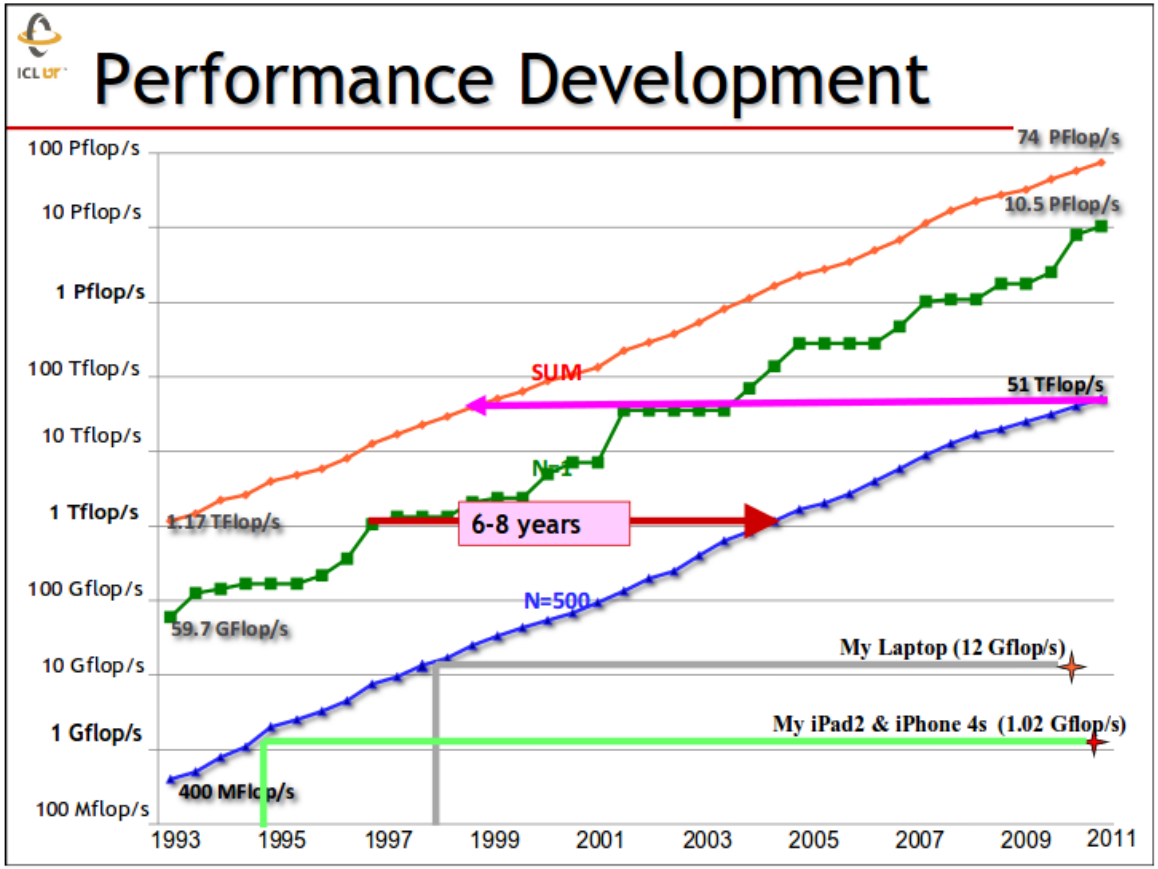
- If $n=64$, this implies an overall reduction in flops of ~16 million*

*Six-months is reduced to 1 s

OSTP Briefing, 4 May 2004

source: David Keyes, prof. of applied math., Columbia

Evolution des performances brutes



Parallélisme existant / *gratuit*

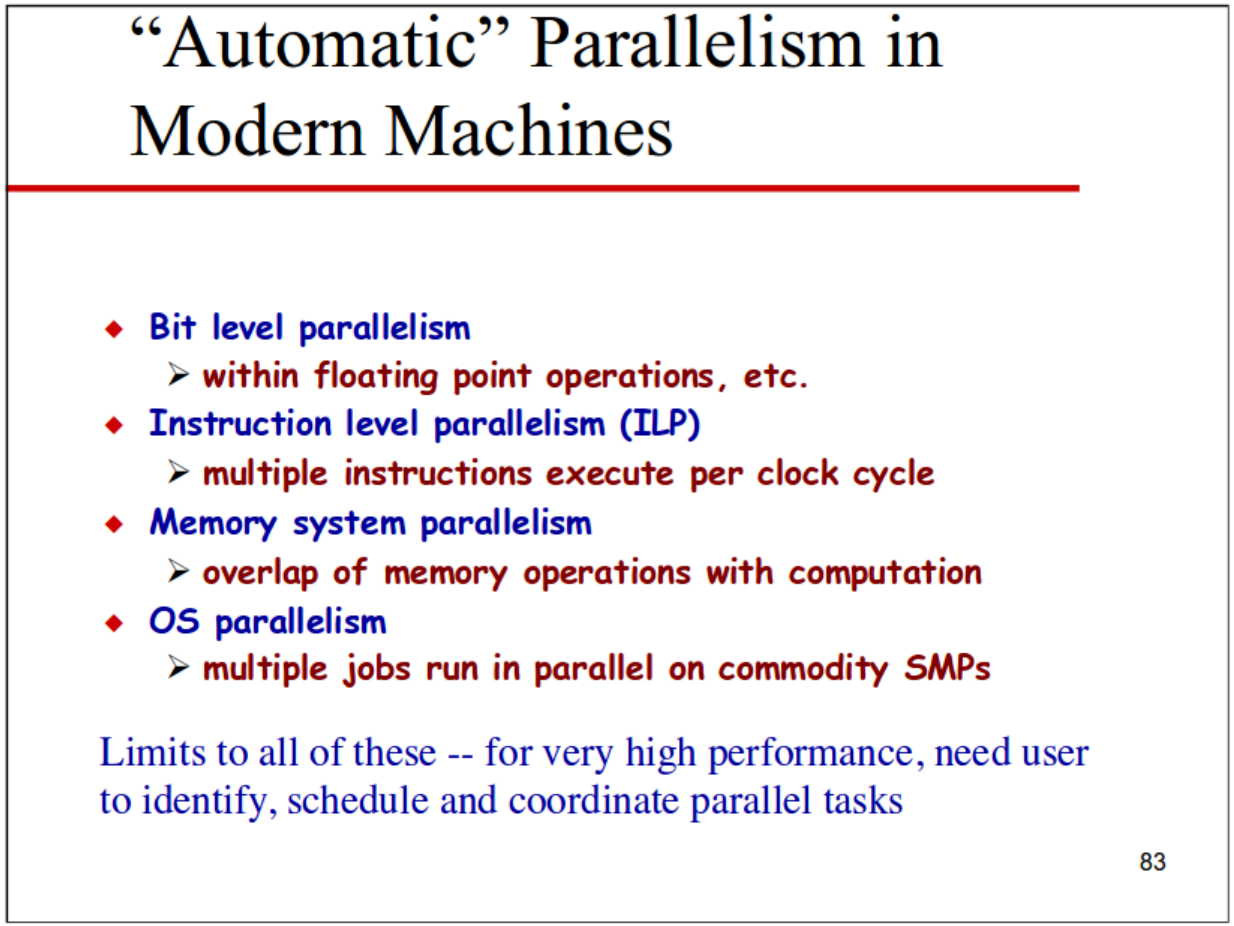


Figure: source: Scientific Computing for engineers, CS594, J. Dongarra

Parallélisme - mots clés

- **scalable speed-up:** Relative reduction of execution time of a fixed size workload through parallel execution

$$\text{Speedup} = \frac{\text{execution\_time\_on\_1\_processor}}{\text{execution\_time\_on\_N\_processor}}$$

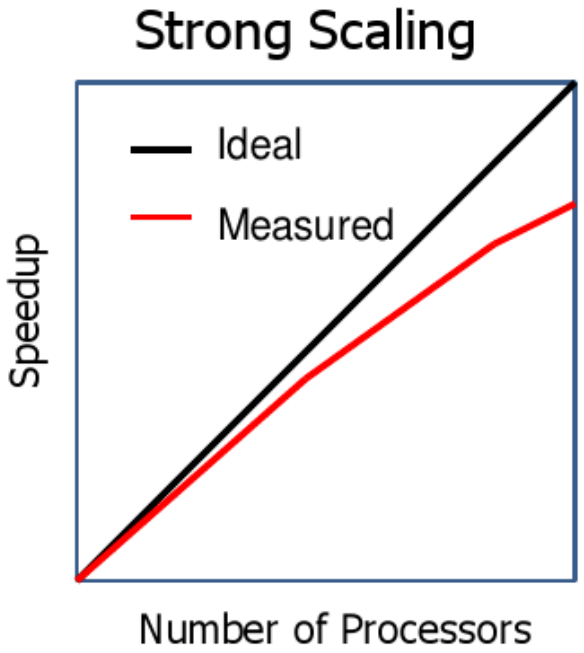
  ideallement: N

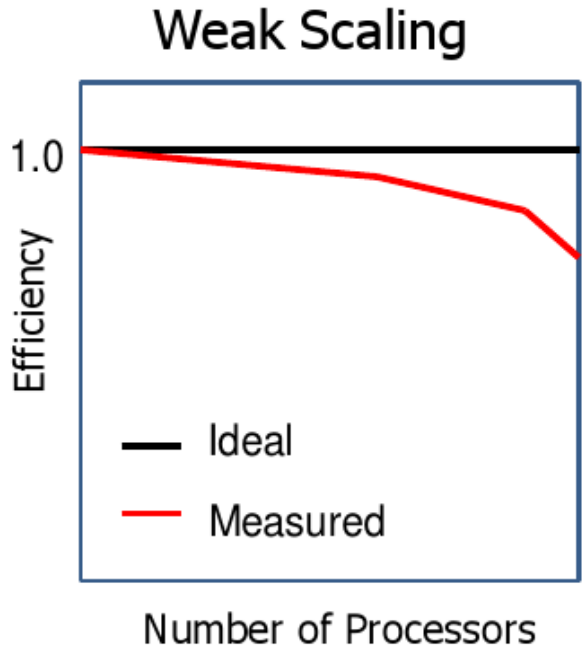
- **scalable efficiency:** Ratio of the actual performance to the best possible performance.

$$\text{Efficiency} = \frac{\text{execution\_time\_on\_1\_processor}}{\text{execution\_time\_on\_N\_processor} \times N}$$

  ideallement: 100%

Parallélisme - *weak / strong scaling*





- Une application / un algorithme utilise-t-elle bien les ressources de calcul de mon cluster ?
- **Weak scaling:** If the problem size increases in proportion to the number of processors, the execution time is constant. If you want to run larger calculations, you are looking for weak scaling.
- **Strong scaling:** For a given size problem, the time to execute is inversely proportional to the number of processors used. If you want to get your answers faster, you want a strong scaling program.

Qu'est ce qu'un super-calculateur ?

La machine CURIE hébergée au TGCC de Bruyères-le-Châtel

Qu'est ce qu'un super-calculateur ?

La machine CURIE hébergée au TGCC de Bruyères-le-Châtel

Qu'est ce qu'un super-calculateur ?

La machine CURIE hébergée au TGCC de Bruyères-le-Châtel

Qu'est ce qu'un super-calculateur ?

- Ce sont des calculateurs dont la puissance de calcul est proche des limites de la technologie contemporaine
- C'est une infra-structure complexe: occupe souvent un bâtiment entier, consommation électrique ~ qq MW à qq 10MW; refroidissement/climatisation très important
- assemblage très spécifique de composants / matériels informatiques; petit nombre de vendeurs (Cray, IBM, HP, DELL, SGI, Intel, Bull, ...)
- Utilisation originale: le calcul scientifique

Qu'est ce qu'un super-calculateur ?

Définitions:

- **Supercomputer:** *A computing system exhibiting high-end performance capabilities and resource capacities within practical constraints of technology, cost, power, and reliability.* Thomas Sterling, 2007.
- **Supercomputer:** *a large very fast mainframe used especially for scientific computations.* Merriam-Webster Online.
- **Supercomputer:** *any of a class of extremely powerful computers. The term is commonly applied to the fastest high-performance systems available at any given time. Such computers are used primarily for scientific and engineering work requiring exceedingly high-speed computations.* Encyclopedia Britannica Online.

Qu'est ce qu'un super-calculateur ?

Survol historique

- CRAY-1: 1976, 80MHz, 64-bit/data, 24-bit/adress, vector register file, 160 MIPS, 250 MFLOPS, 8MB RAM, 5.5 tonnes, ~ 200-kW (cooling included)
- les premiers supercalculateurs sont des machines utilisant du matériel spécialement conçu pour cette utilisation; les supercalculateurs des années 1980 et les ordinateurs personnels ont très peu de choses en commun
- depuis la fin des années 90, la tendance s'inverse; on utilise de plus en plus de composants commerciaux (*Off-the-shelf*)

source: http://en.wikipedia.org/wiki/Supercomputer

Qu'est ce qu'un super-calculateur ?

Leading technology paths (to exascale) using TOP500 ranks (https://www.top500.org/, Nov. 2012)

- **Multicore:** Maintain complex cores, and replicate (x86, SPARC, Power7) (#3, 6, and 10)
- **Manycore/Embedded:** Use many simpler, low power cores from embedded (IBM BlueGene) (#2, 4, 5 and 9)
- **GPU/MIC/Accelerator:** Use highly specialized processors from gaming/graphics market space (NVidia Fermi, Cell, Intel Phi (MIC)), (# 1, 7, and 8)

Qu'est ce qu'un super-calculateur ?

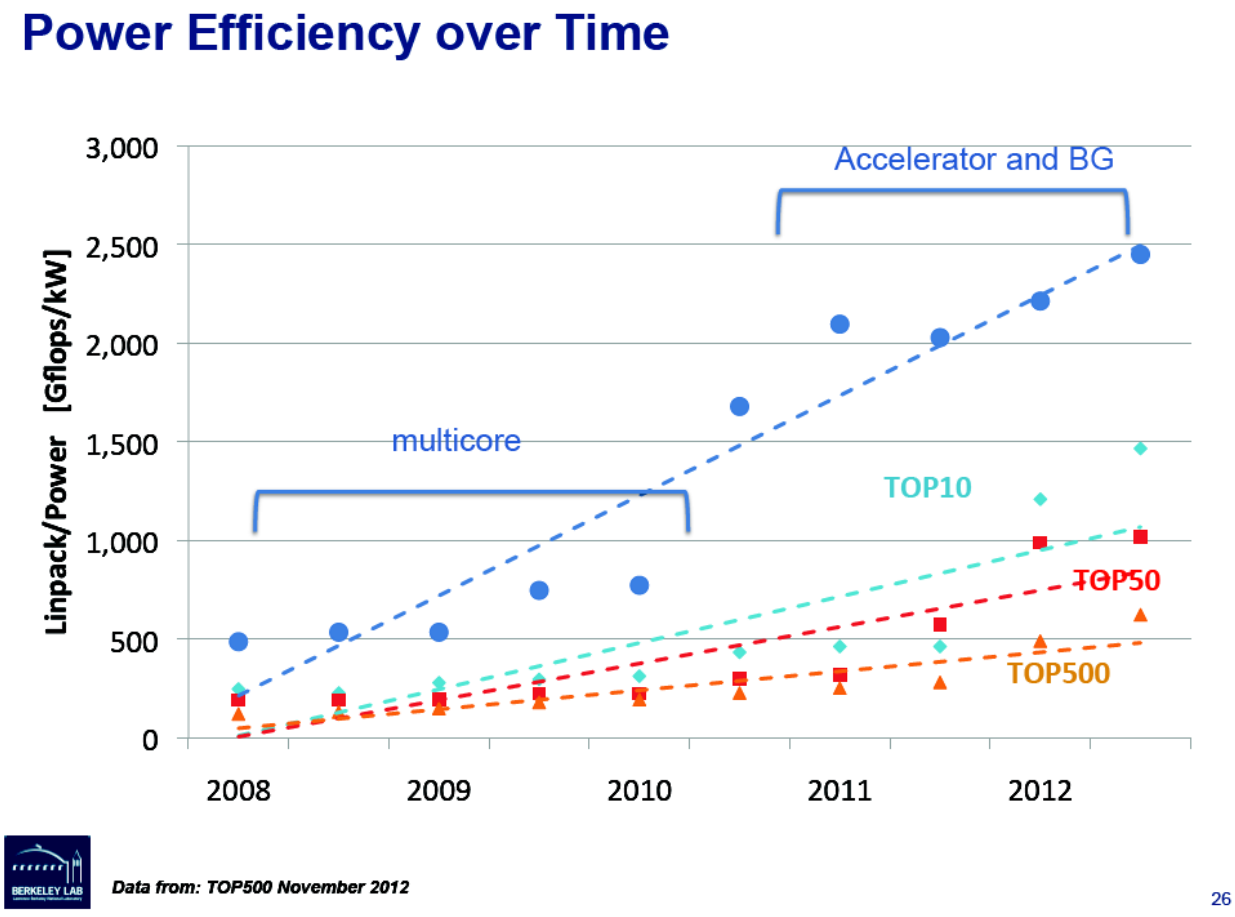


Figure: Horst Simon, LBNL

Qu'est ce qu'un super-calculateur ?

Parallel processing models (not anymore ?)-used in supercomputer

- **Communicating Sequential Processing** - MPI
- **Shared memory multiple thread** - OpenMP / pthread
- **SIMD** - vector instruction, lowest level
- **Accelerators** - GPU / XeonPhi / (FPGA)
- Alternative models
  - Vector machines: Hardware execution of value sequences to exploit pipelining
  - Systolic: An interconnection of basic arithmetic units to match algorithm
  - Data Flow: Data precedent constraint self-synchronizing fine grain execution units supporting functional (single assignment) execution

Qu'est ce qu'un super-calculateur ?

Linux Cluster Overview by example at LLNL

Comment mesurer / évaluer les performances un programme (parallèle) ?

Quelles métriques utiliser ?

- Liste des supercalculateurs les plus puissants: TOP500
- Ca peut dépendre du type d'algorithme:
  - Algorithme dit *compute bound* (ex: tri): FLOPS
  - Algorithme dit *memory bound* (ex: tri): bande-passante mémoire
- Métriques
  - une quantité mesurable représentant le taux d'exécution d'une tâche
  - Instructions par seconde
  - Puissance électrique (1 MW ~ 1 M€)
  - Performance par Watt (Green500)

Comment mesurer / évaluer les performances un programme (parallèle) ?

Quelles métriques utiliser ?

- Utiliser un benchmark
  - classement TOP500: GFLOPS obtenu sur l'execution du LINPACK
  - benchmarks parallèles:
    - NPB - NAS parallel benchmark from NASA Advanced Supercomputing Division
    - hpcc
    - Parboil
    - Rodinia (application sur architectures hétérogènes - GPU)
    - SHOC (application sur architectures hétérogènes - GPU)
    - HPCtoolkit
    - HOMB: solveur de Laplace (hybrid MPI / OpenMP)
  - benchmarks sur les IO parallèles (*filesytem / hard drive*): IOR, ParallelIO, ...

Supercomputers / Mesure de performance / Benchmarks - Mini-apps

- Lawrence Livermore National Lab mini-apps

LULESH	Explicit lagrangian shock hydrodynamics on unstructured mesh representation
AMG2013	Algebraic Multi Grid
Mulard	Unstructured mesh, finite element, implicit multigroup radiation diffusion
UMT	Unstructured mesh Transport
MCB	Monte Carlo Particle Transport
LKS	Suite of kernels in a unified framework for testing compilers SIMD and threading
DRIP	2D interpolation on tabular data
LUAU3D	Material advection on an unstructured hexahedral mesh

- MANTEVO (Sandia National Lab)

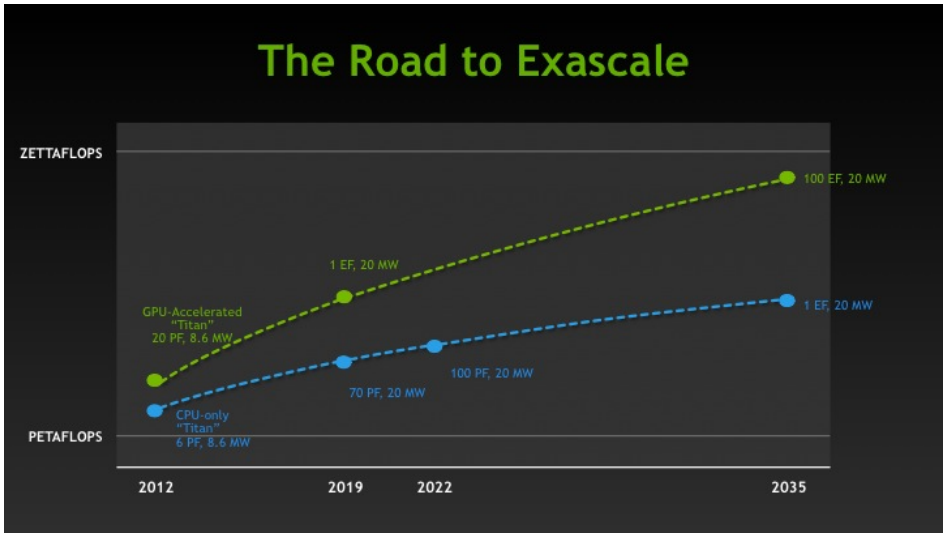
Outils d'aide à l'analyse / à la mesure de performance:

- *temps d'éxecution*: `time`, *gettimeofday*
- *Profiling*: `gprof`, perf, PAPI
- *Tracing*:
  TAU,
  scoreP/ scalasca/ cube
  hpctoolkit
  Intel VTune
  mpiP (lightweight, no-GUI), tuto mpiP

- Voir le tableau récapitulatif sur le site de LLNL
- MPIBlib: P2P, collective MPI communication benchmark

Exascale : back in the day...

- At SC2011, Nvidia CEO talk (a bit optimistic):



- Document The International Exascale Software Project Roadmap by Dongarra et al. (DOI:10.1177/1094342010391989)
  1. Make a thorough assessment of needs, issues and strategies,
  2. Develop a coordinated software roadmap,
  3. Encourage and facilitate collaboration in education and training.
- **Artificial Intelligence (AI) was not even in the scope...**

Exascale race: technological challenge(s)

Exascale is about...

- ... reaching exaflops = 10^{18} flop/s
  - 1.3 exaflops of aggregate peak flops
  - ~ 8 PBytes of RAM
- **Building a computing ecosystem:** from applications, system software, hardware technologies, and architectures
- Doing usefull work : HPC simulation ? AI ? both !
- timeframe : ~ 2021 - 2022

Summit

current #1 @top500

Post-K (Japan, 2021 ?)

Many challenges:

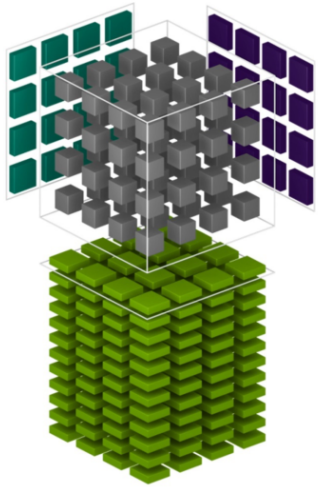
- **Most challenging constraint:** fitting the **electrical power envelop** (P in $[20-40]$ MWatts)
- develop new applications / adapt old ones
- system software, application software stacks
- hardware technologies (**interconnect, storage, processor architectures,...**)

Exascale race: technological challenge(s)

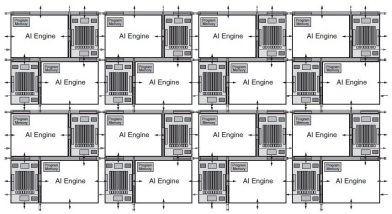
...also about building an economic ecosystem

- New architectures designed to address **several markets** : HPC, AI, IoT, near-sensor computing, automotive, ...
- **Hardware vendors already designing/optimizing new architectures for AI** (always back and forth between general purpose and application specific):e.g
  - Nvidia (Tensor Core),
  - Xilinx (Alveo / Versal),
  - Intel (BFLOAT16, for future CooperLake), ...
- **Semantic shift: HPC = simulation + AI**
- **Cost of designing a new chip skyrocketting, ...**

Nvidia Tensor Core, Volta (2017)



Xilinx AI Engine array (2019)

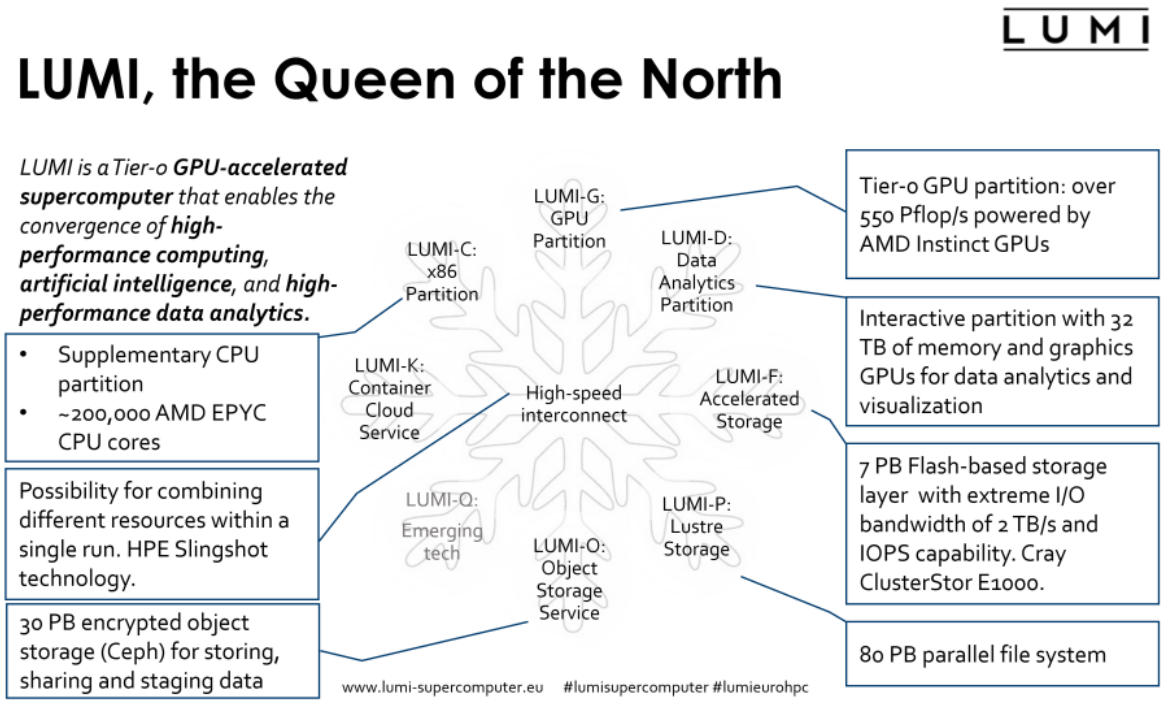


Pre-Exascale machines - architecture diversity !

- **US**: Summit , Sierra ⇒ mostly OpenPower (IBM P9 + Nvidia V100), GPU-based architecture, #1 and #2 @top500; exascale machines announced
  - Aurora (Argonne NL, 2022): Intel Xe GPU
  - Frontier (Oak Ridge NL, 2021 ?): AMD EPYC + Radeon GPU
- **China:** 3 machines
  - Phytium FT2000/64 ARM chips + Matrix2000 GPDSP accelerators ⇒ #4 @top500, Tianhe-2A, 61 PFlops
  - 260-core Shenwei, **homegrow technology** hardware + software (C++/fortran compiler + OpenACC) ⇒ #3 @top500 , Sunway TaihuLght, 93 PFlops
  - Dhyana, AMD-licenced x86 multicore (300 M$!), identical to AMD EPYC
- **Japan:** **Post K**(Fujitsu, ARM, RIKEN) A64FX ARM (**home grown**, started in 2014, 900 M$), GPU, etc ...
- **Europe** : was lagging behind but new organization EuroHPC (2019), EC H2020 budget (~ 500 M€)
  home grown ARM and RISC-V architecture, early stage

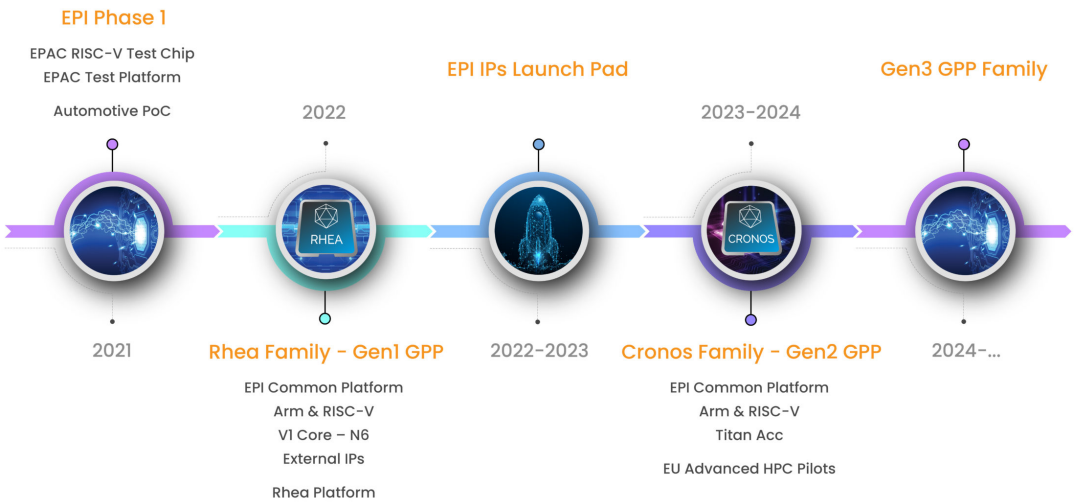
Largest supercomputer in Europe (2021)

- LUMI, 550 PFlops (current world #1 is Japanese Fugaku, 442 PFlops)
- AMD GPU Instinct MI100 (Nov. 2020)



European Processor Initiative (EPI)

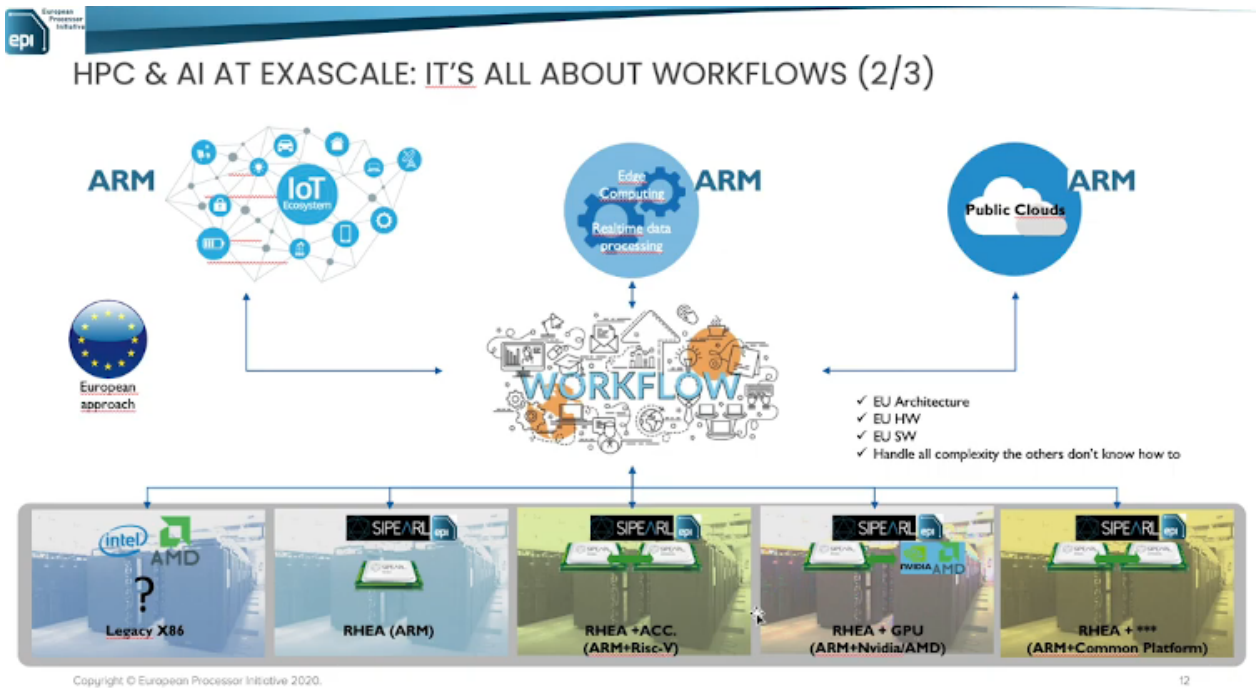
- project coordinated by ATOS/Bull:
  - hardware : an ARM-based procssor for the first european exascale machine, EPAC is a RISC-V accelerator architecture
  - software : a clang-based compiler for RISC-V Vector extension, ...
  - design/manufacture/sell : SiPearl, a fabless company for embedded, IoT, automotive, HPC, ...
- started in Dec. 2018



https://www.european-processor-initiative.eu/project/epi/

European Processor Initiative (EPI)

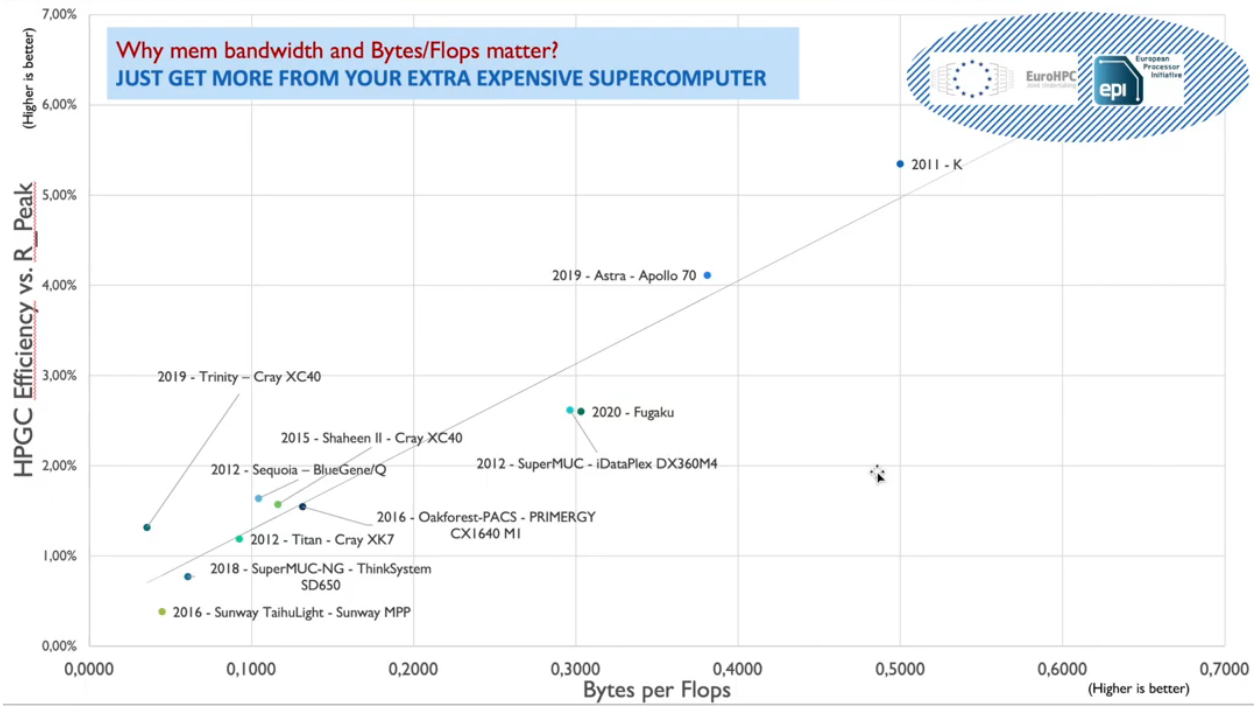
- project coordinated by ATOS/Bull:
  - hardware : an ARM-based procssor for the first european exascale machine, EPAC is a RISC-V accelerator architecture
  - software : a clang-based compiler for RISC-V Vector extension, ...
  - design/manufacture/sell : SiPearl, a fabless company for embedded, IoT, automotive, HPC, ...
- started in Dec. 2018



https://www.european-processor-initiative.eu/project/epi/

European Processor Initiative (EPI)

- project coordinated by ATOS/Bull:
  - hardware : an ARM-based procssor for the first european exascale machine, EPAC is a RISC-V accelerator architecture
  - software : a clang-based compiler for RISC-V Vector extension, ...
  - design/manufacture/sell : SiPearl, a fabless company for embedded, IoT, automotive, HPC, ...
- started in Dec. 2018



https://www.european-processor-initiative.eu/project/epi/

EuroHPC

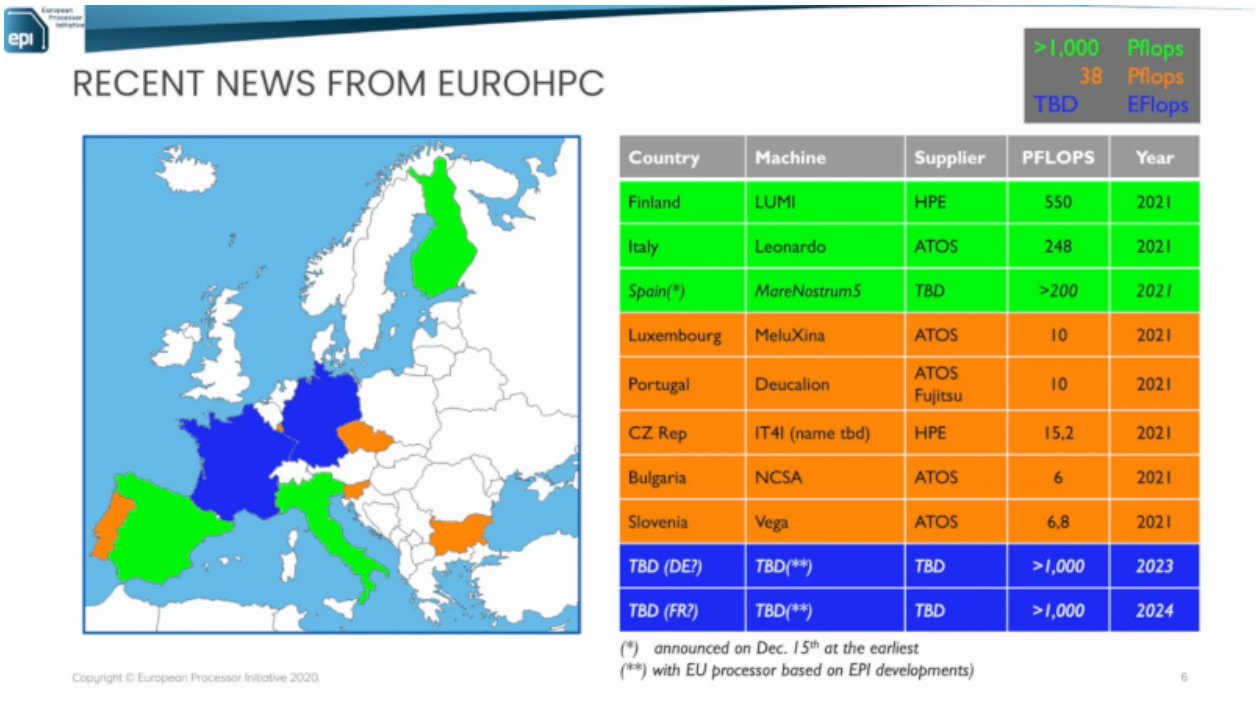
https://eurohpc-ju.europa.eu/, created in 2018; drives how and where supercomputing resources should be settled in Europe.

EuroHPC

https://eurohpc-ju.europa.eu/, created in 2018; drives how and where supercomputing resources should be settled in Europe.

EuroHPC

https://eurohpc-ju.europa.eu/, created in 2018; drives how and where supercomputing resources should be settled in Europe.



RECENT NEWS FROM EUROHPC

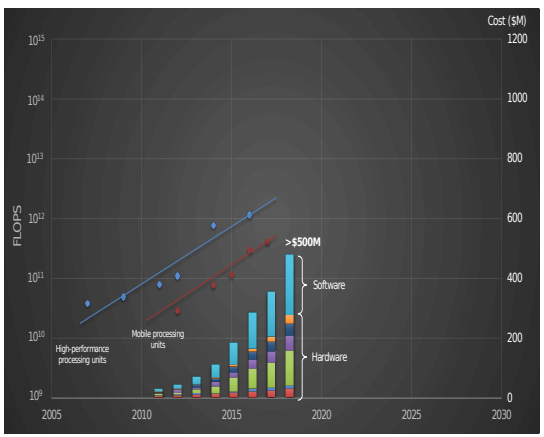
>1,000 Pflops
38 Pflops
TBD EFlops

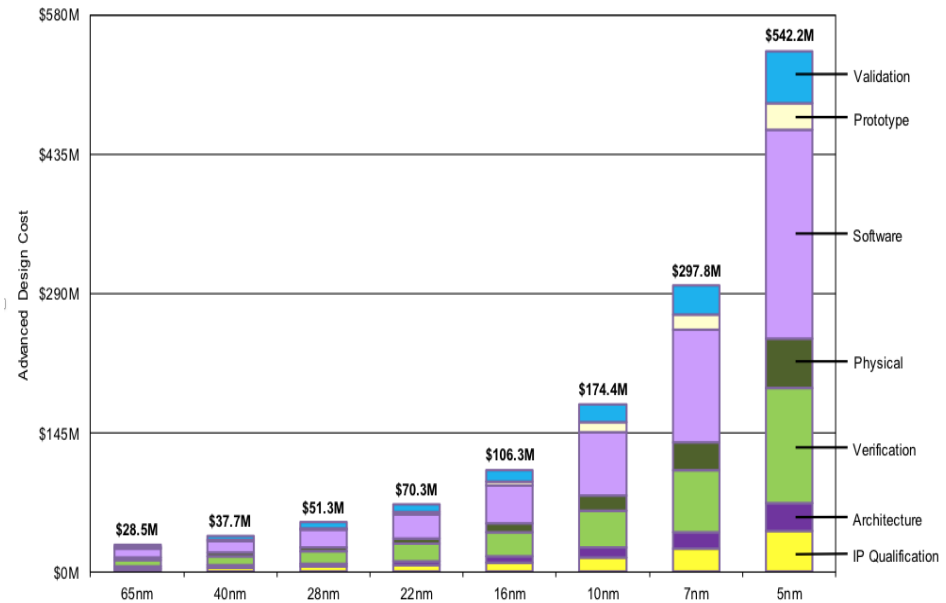
Country	Machine	Supplier	PFLOPS	Year
Finland	LUMI	HPE	550	2021
Italy	Leonardo	ATOS	248	2021
Spain(*)	*MareNostrum5*	*TBD*	*>200*	*2021*
Luxembourg	MeluXina	ATOS	10	2021
Portugal	Deucalion	ATOS Fujitsu	10	2021
CZ Rep	IT4I (name tbd)	HPE	15,2	2021
Bulgaria	NCSA	ATOS	6	2021
Slovenia	Vega	ATOS	6,8	2021
TBD (DE?)	*TBD(**)*	*TBD*	*>1,000*	*2023*
TBD (FR?)	*TBD(**)*	*TBD*	*>1,000*	*2024*

(*) announced on Dec. 15th at the earliest
(**) with EU processor based on EPI developments)

Beyond Moore's Law ...

- Semiconductor sector still growes faster than GDP (PIB)
- Current chip makers (Intel, NVidia, Xilinx, ...) increasingly focus on specific applications : AI,... not necessarily HPC driven anymore
- **Skyrocketing cost of chip development limits hardware innovation.**
- June 2017 : **DARPA Electronics Resurgence Initiative** : 5-year 1.5 B$ investment for new chip architectures, IC design and materials and integration. ⇒ **reduce costs of chip design**

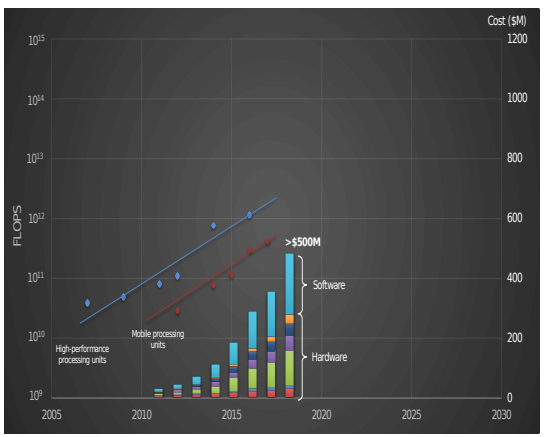


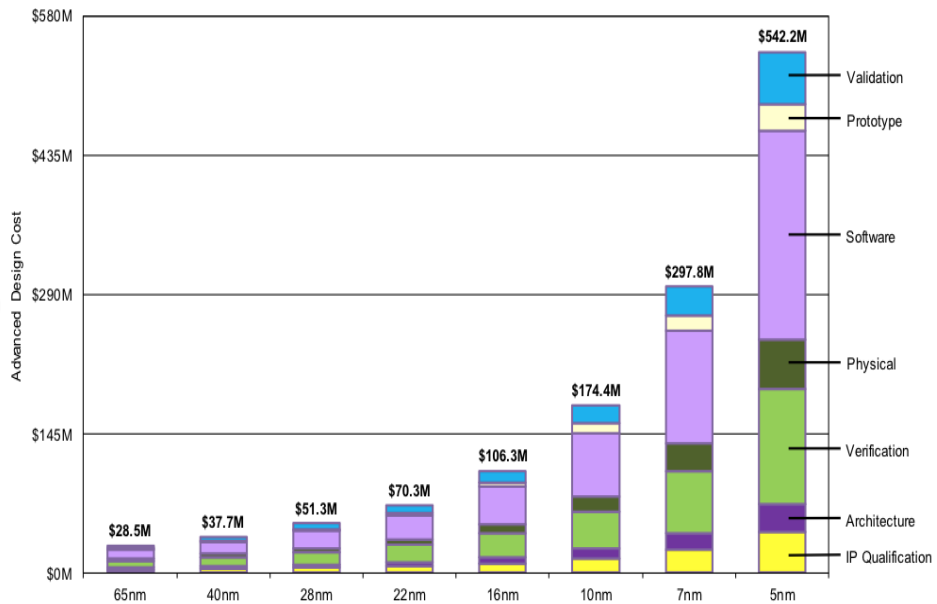


https://www.enterprisetech.com/2018/07/24/darpa-effort-pushes-beyond-moores-law

Beyond Moore's Law ...

- **The cost of chip making... needs huge revenue to justify these expenses**
- Nvidia Volta/Turing (12 nm), next-gen in 7 nm
- AMD Vega / ZEN2 (2019): 7nm
- Xilinx Versal FPGA (2019): 7nm
- Fujitsu ARM 64 bit for HPC (2019-2020): 7nm
- **nothing beyond 3nm (if reachable) ? ⇒ 3D integration package**

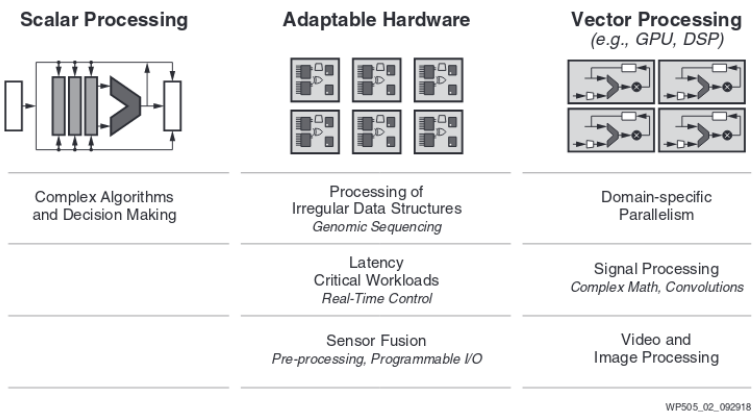




https://www.enterprisetech.com/2018/07/24/darpa-effort-pushes-beyond-moores-law

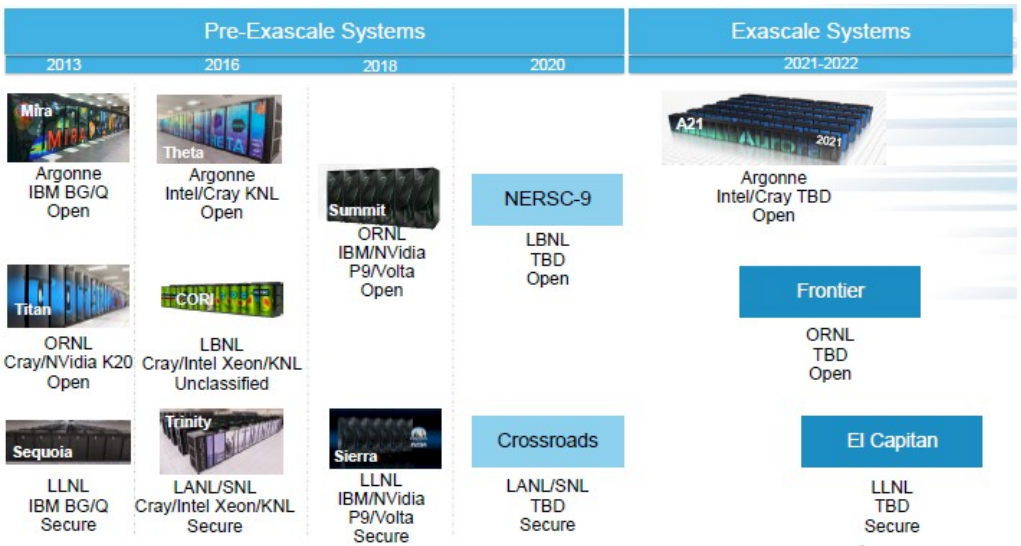
Beyond Moore's Law ...

- **What's next ?**
- **Rethinking hardware architecture design:**
  - not necessarily increasing die size
  - 2.5D or 3D packages integration
  - Open source hardware : e.g. RISC-V (young tech., 2015, just a co-processor ?)
    ⇒ good for startup innovation, prototyping, many hardware *flavors*, on-chip heterogenous computing
    ⇒ fast growing markets (AI / IoT, automotive)
  - many different end markets: IoT, automotive, data centers (cloud, AI)...
  - **More hybrid architectures ?** see e.g. **Xilinx versal (2019) all-in-one**



Exascale roadmap in the US

- CORAL program : **Hardware** (Summit, Sierra) + **Software** (programing models : OpenMP/OpenAcc, Kokkos, ...)
- CORAL-2 ⇒ **exascale** (cognitive simulation systems ?)
- Costs:
  - Titan (2012) ~ 100M\$
  - Summit (2018) ~ 200M\$
  - **futur exascale system : between** ~ 400M\$ **and** ~ 600M\$

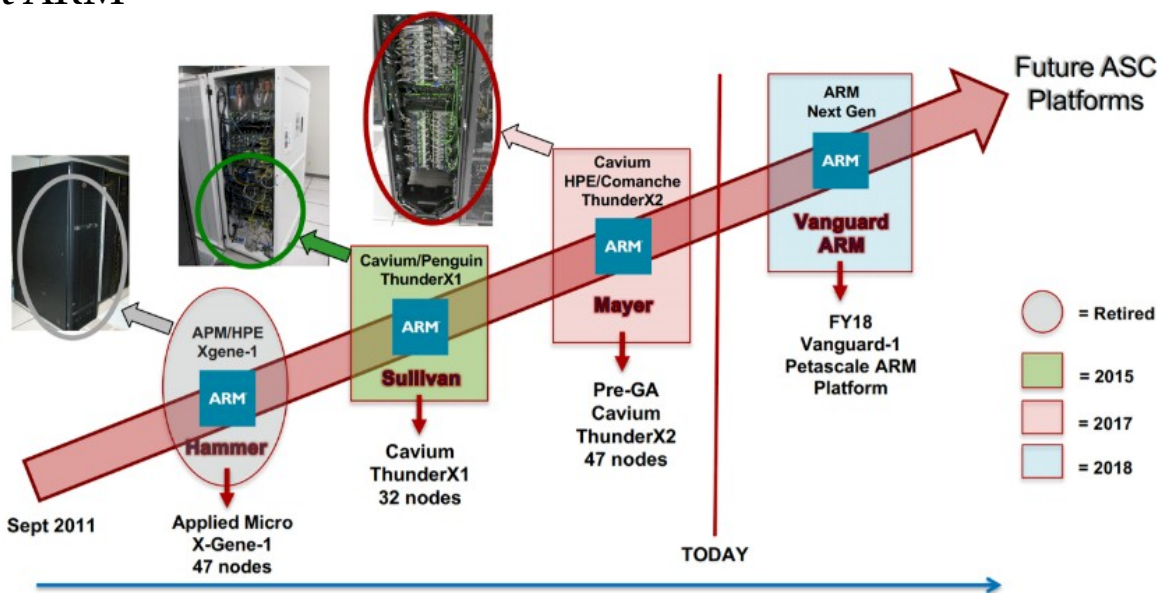


https://www.nextplatform.com/2018/04/09/bidders-off-and-running-after-1-8-billion-doe-exascale-supercomputer-deals/

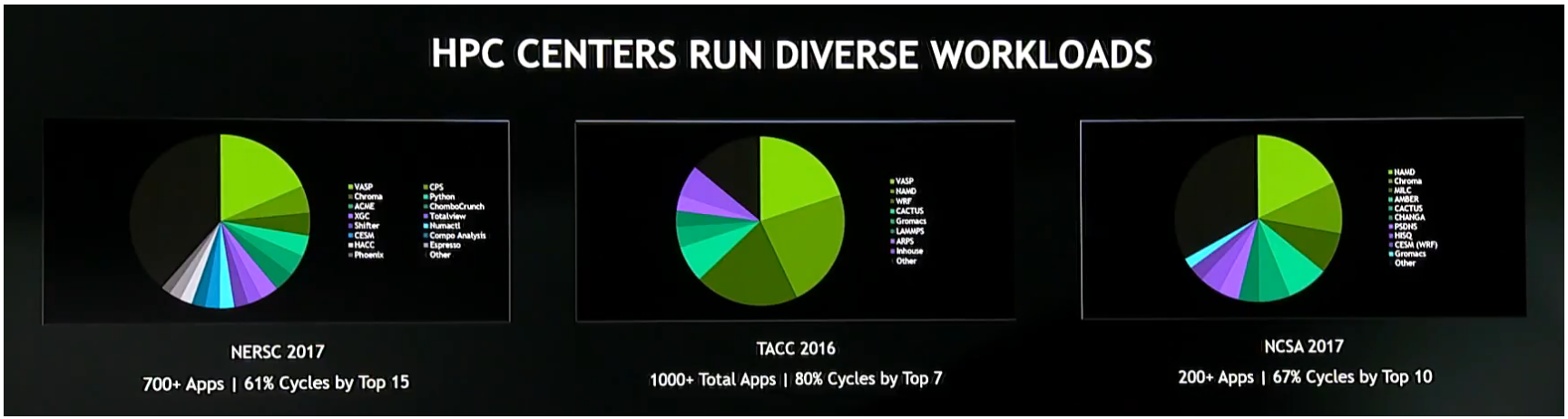
https://www.nextplatform.com/2018/03/06/roadmap-ahead-exascale-hpc-us/

Multicore architectures: X86, ARM, ...

- **INTEL X86**: skylake (24 cores), cascadelake (Deep Learning Boost, MKL-DNN), ...
- **AMD EPYC** (64 cores, very promising, 7nm, 2019)
- **ARM** (32 cores) : ThunderX2 at Sandia NL, LANL, Japan, Europe, ...
  - `Astra` (SANDIA NL), 2592 nodes, dual-socket, 28-cores ThunderX2 64 bit ARM



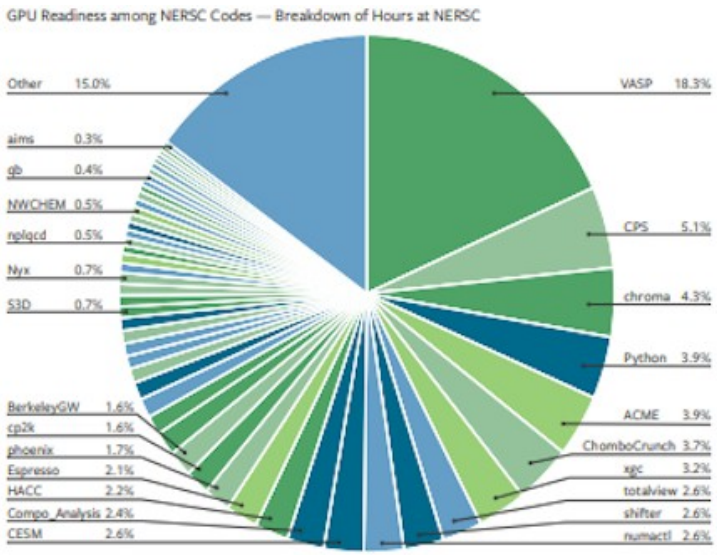
HPC centers workload per scientific domain



SC18: NVIDIA CEO talk

Software challenge(s) for (pre-)exascale machines

GPU readiness: About chosing architecture for NERSC-9 machine (Pre-exascale @LBNL), probably AMD EPYC + Nvidia GPUs (2020)



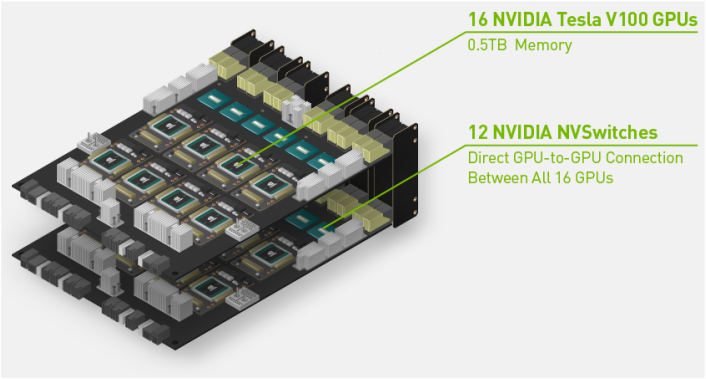
GPU Status	Description	Fraction
Enabled	Most features are ported and performant.	32%
Kernels	Ports of some kernels have been documented.	10%
Proxy	Kernels in related codes have been ported.	19%
Unlikely	A GPU port would require major effort.	14%
Unknown	GPU readiness cannot be assessed at this time.	25%

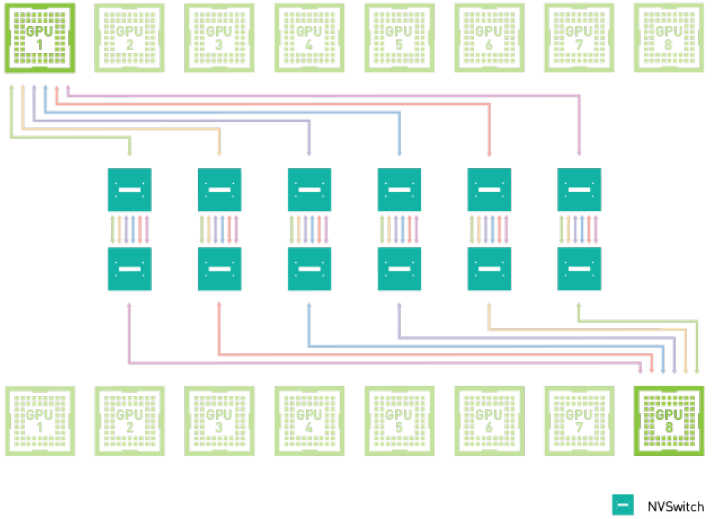
New fast interconnect by CRAY: named SHASTA, enabling new applications, real-time image processing data from telescopes (e.g. already existing ASTRON for Fast Radio Burst, ...)

https://www.nextplatform.com/2018/10/30/berkeley-lab-first-in-line-for-cray-shasta-supercomputers/

Nvidia roadmap for HPC and AI - hardware

HGX-2 - heavy fat GPU-based node



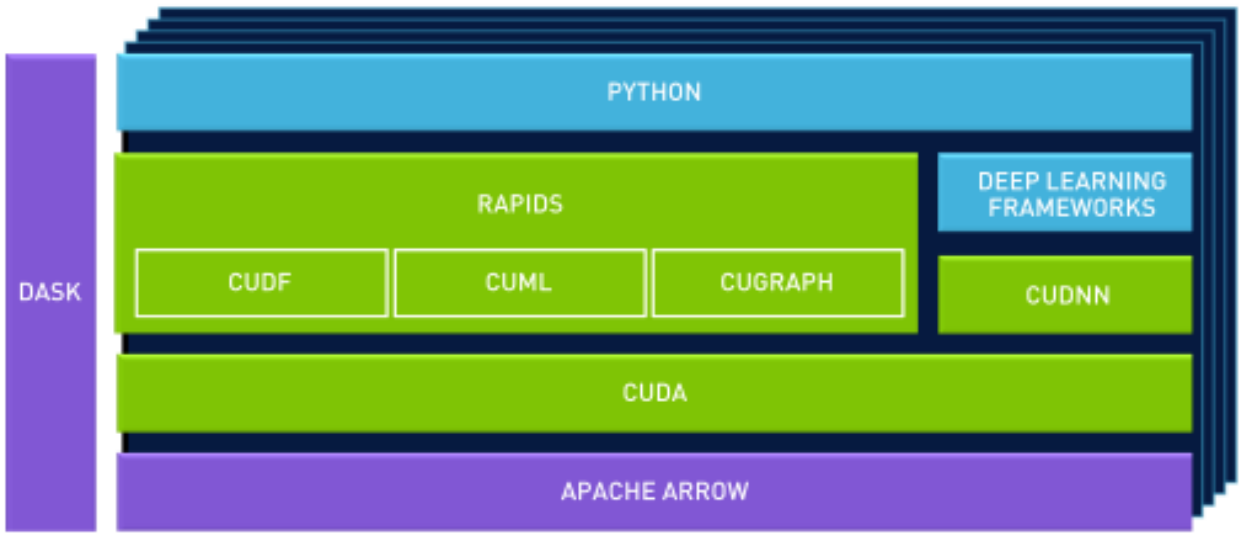


- Mostly for AI applications in the cloud, expensive (400 k$)
- If Summit (#1 @ top500, 27 000 GPU) was assembled with HGX2 => 675 M$ (for the GPUs only !)

Nvidia roadmap for HPC and AI - software

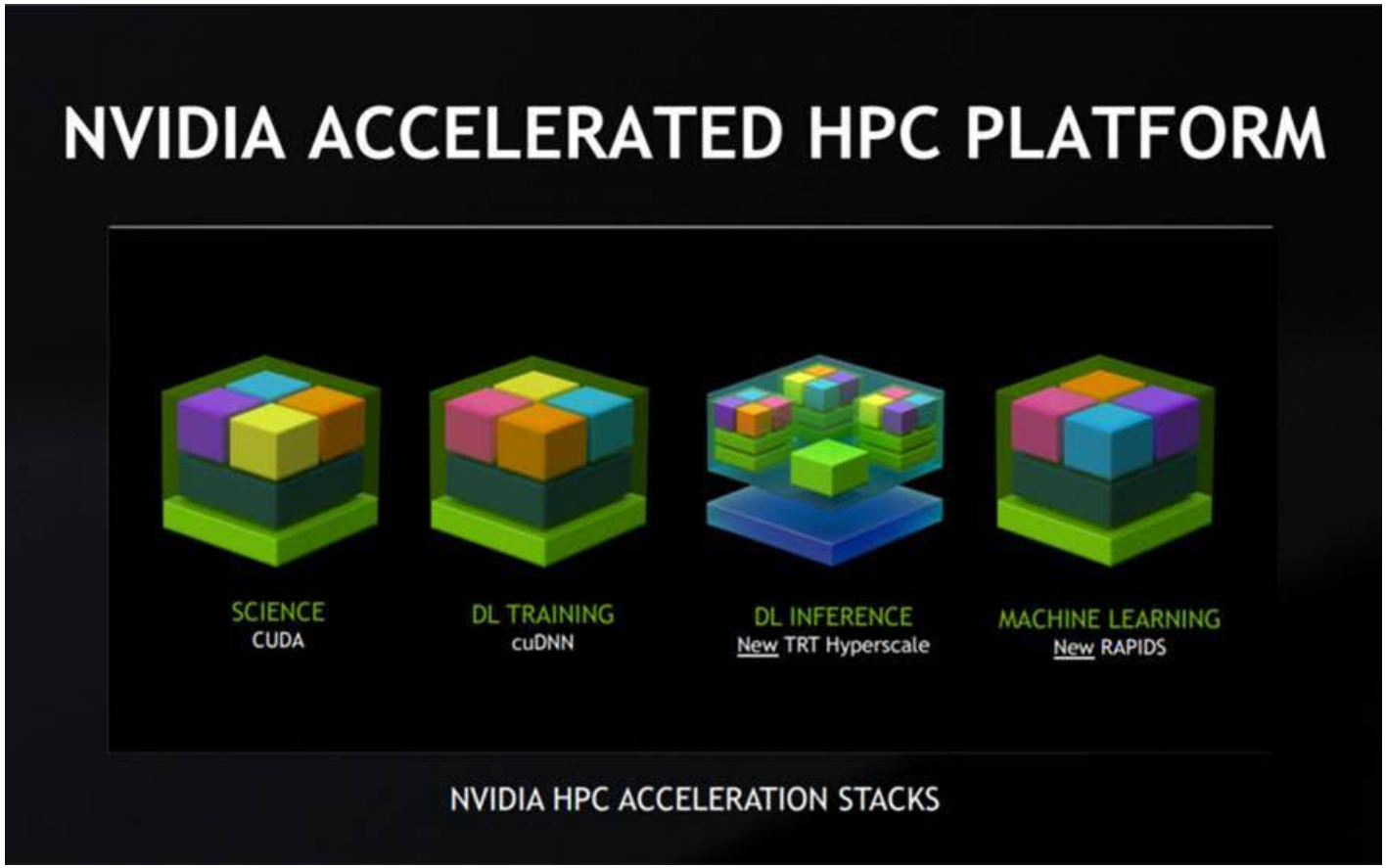
Semantic shift: HPC = simulation + AI

Machine Learning to Deep Learning: All on GPU



Nvidia roadmap for HPC and AI - software

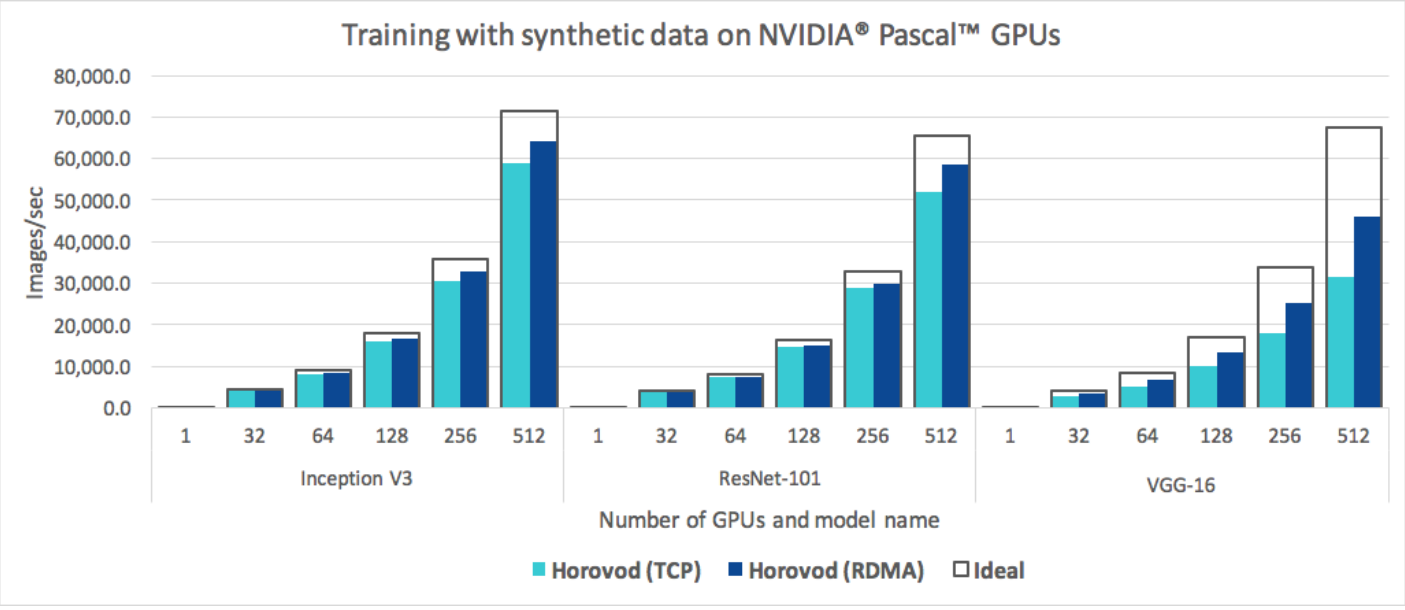
Semantic shift: HPC = simulation + AI



Does AI need HPC hardware ?

Distributed training on modern HPC hardware (optimizing `MPI_Allreduce`)

- exponential growth, from single-node/single GPU to modern HPC cluster



horovod-mpi

Cloud - HPC convergence

- **Cloud : externalize workload (mostly services) in data center**
- **regular cloud**: relies on virtualization, the customer instanciates virtual machines ⇒ many VM fight for hardware resources
- **bare-metal cloud** is just dedicated hardware resource; the customer pay for hardware nodes and do what ever he wants with (deploy docker images, ...)
- 2018, from a user survey [1]
  - regular cloud : 75 %
  - bare-metal cloud : 25 % (was 11 % in 2016)

[1] https://www.nextplatform.com/2018/09/05/future-clouds-could-be-just-containers-on-bare-metal/

Cloud - HPC convergence

Bare-metal HPC clouds ⇒ alternatives / complement to on-premises clusters and datacenters.
Oracle HPC Cloud offer (announced at SC18)

- Intel Xeon 36-cores running at 3.7 GHz,
- 6.4 TB of NVM-Express drives
- upto a petabyte of block storage
- network bandwidth : 100 Gb/sec and latency of 1.5 microseconds
- **pricing starts at 7.5 cents per core per hour.**
- **4.5 cts per GB / month storage**
- **2.25 $ per GPU/h (Nvidia Volta V100)**

https://www.nextplatform.com/2018/11/13/oracle-puts-together-rdma-bare-metal-for-hpc/

https://www.oracle.com/corporate/pressrelease/oracle-cloud-unveils-hpc-offerings-111218.html

Cloud - HPC convergence

Bare-metal HPC clouds ⇒ alternatives / complement to on-premises clusters and datacenters.

GENCI (French HPC) compute/data center cost estimations (june 2018)

	Supercalculateurs	**Valorisation moyennée** en cts d'une heure/cœur/machine	Valorisation en euros pour 1 Mh
CINES	Bull – Occigen	3,2	31 771 €
IDRIS	IBM – Nœuds larges Ada	2,2	21 761 €
	IBM BG/Q – Turing	0,7	7 254 €
TGCC	Bull – Irene SKL	4,3	43 523 €
	Bull – Irene KNL	1,3	12 903 €

http://www.genci.fr/sites/default/files/Courrier-cadrage-campagne-genci.pdf

Von Neumann architecture

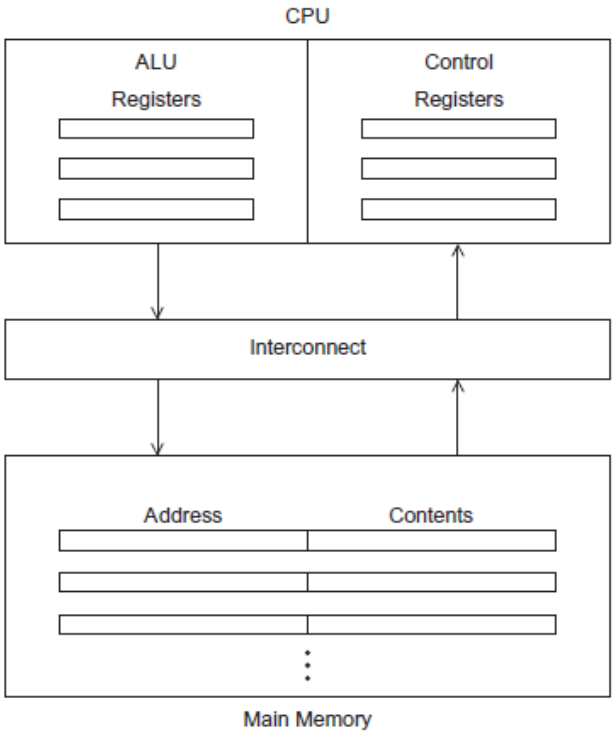


Figure: Peter Pacheco, SanFrancisco U.

Multi-core CPU

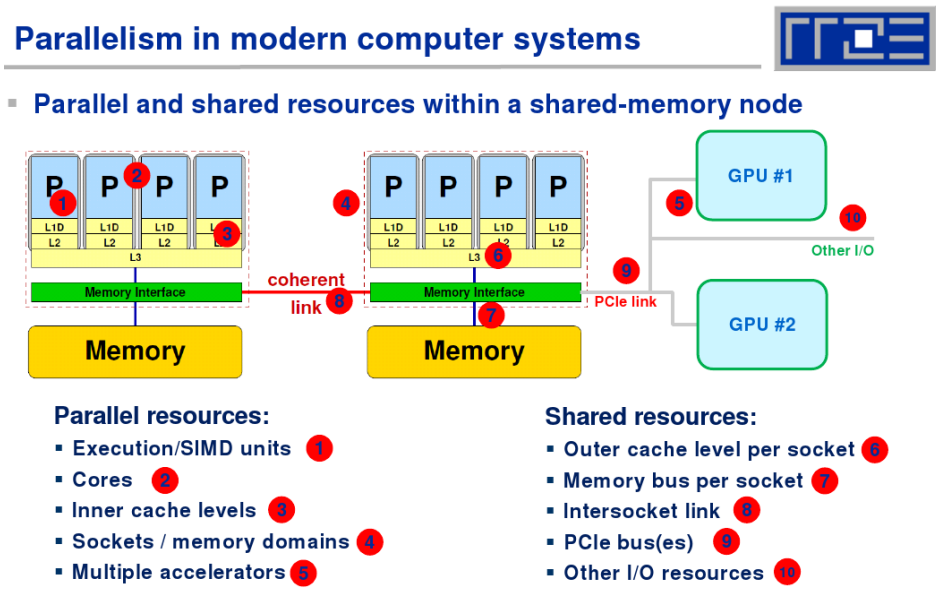


Figure: source: multicore tutorial (SC12) by Georg Hager and Gerhard Wellein

Processor Core Micro architecture

- Execution Pipeline
  - Stages of functionality to process issued instructions
  - Hazards are conflicts with continued execution
  - Forwarding supports closely associated operations exhibiting precedence constraints
- Out of Order Execution
  - Uses reservation stations
  - hides some core latencies and provide fine grain asynchronous operation supporting concurrency
- Branch Prediction
  - Permits computation to proceed at a conditional branch point prior to resolving predicate value
  - Overlaps follow-on computation with predicate resolution
  - Requires roll-back or equivalent to correct false guesses
  - Sometimes follows both paths, and several deep

Hardware: memory hierarchy - low / high latency

- Most programs have a high degree of **locality** in their access
  - spatial locality: accessing things nearby previous accesses
  - temporal locality: reusing an item that was previously accessed
- Main memory (DRAM - off) has high latency compared to on-chip register ⇒ need for intermediate staging area: **cache memory**
- Memory hierarchy tries to exploit locality

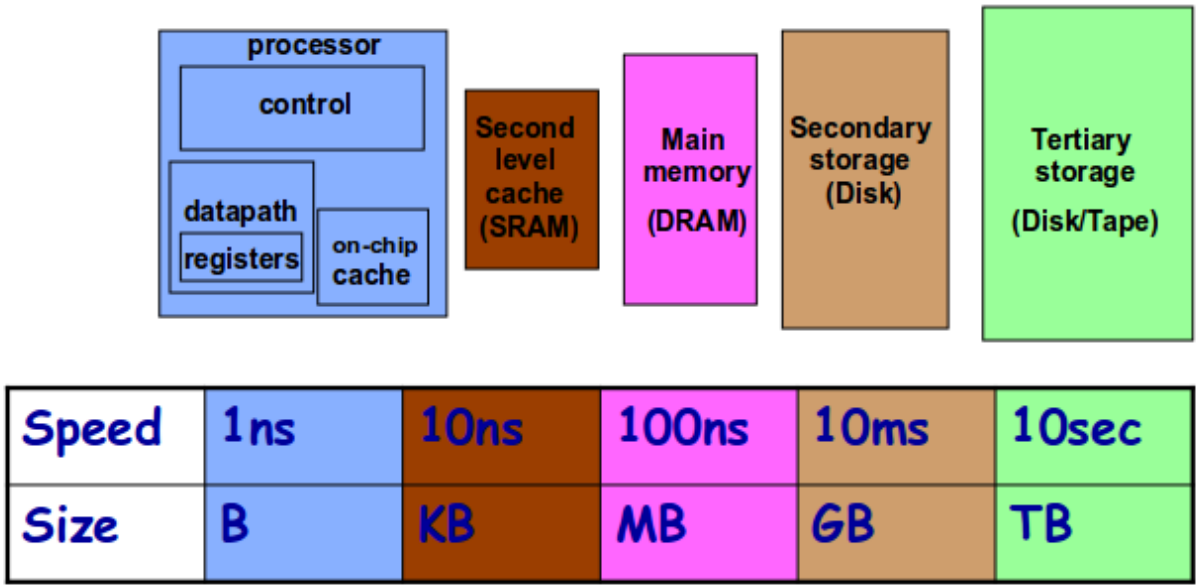


Figure: source: Scientific Computing for engineers, CS594, J. Dongarra

Hardware: memory hierarchy - low / high latency

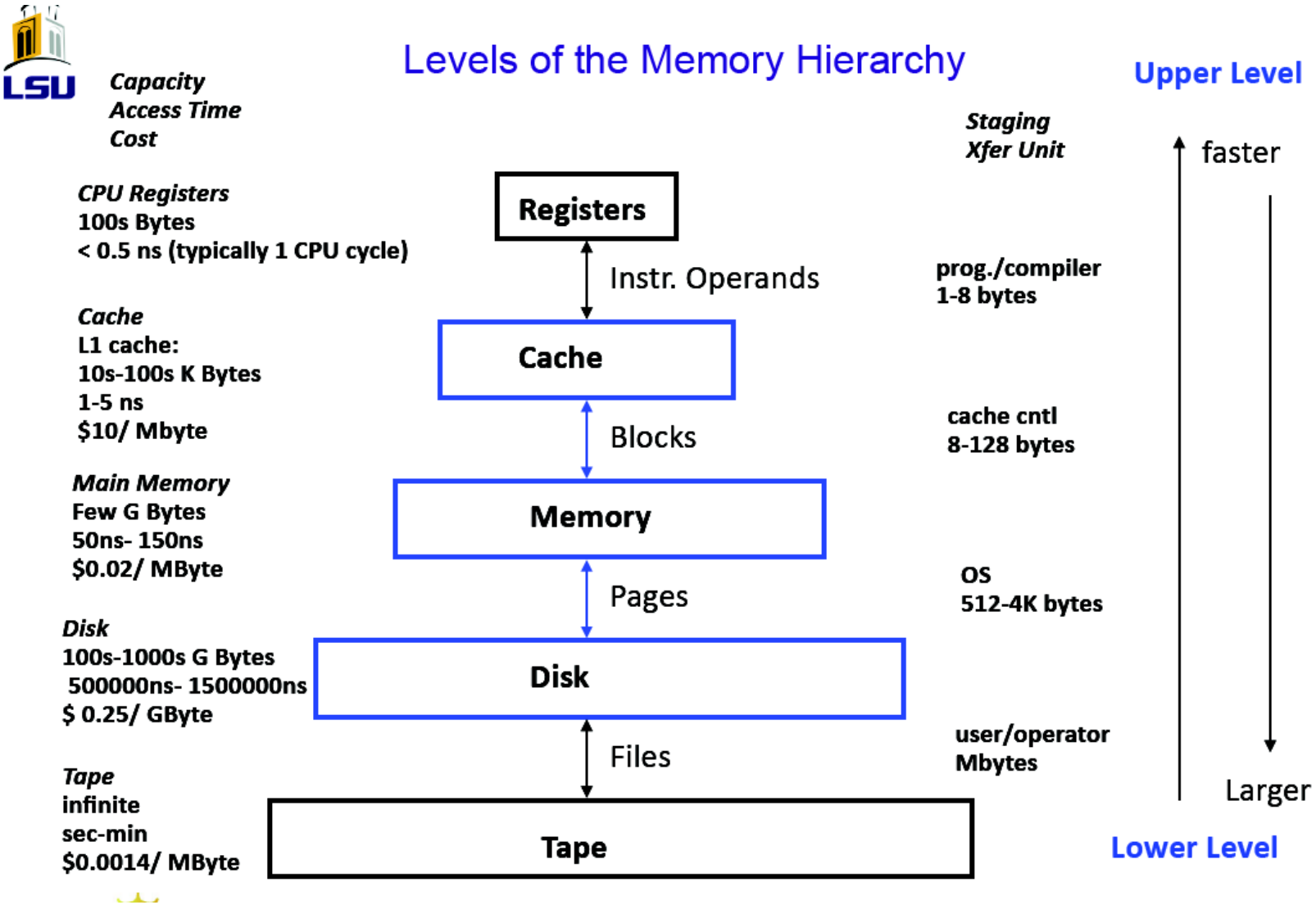


Figure: source: T. Sterling, Louisiana State University, SC12 Tutorial

Hardware: memory hierarchy - low / high latency

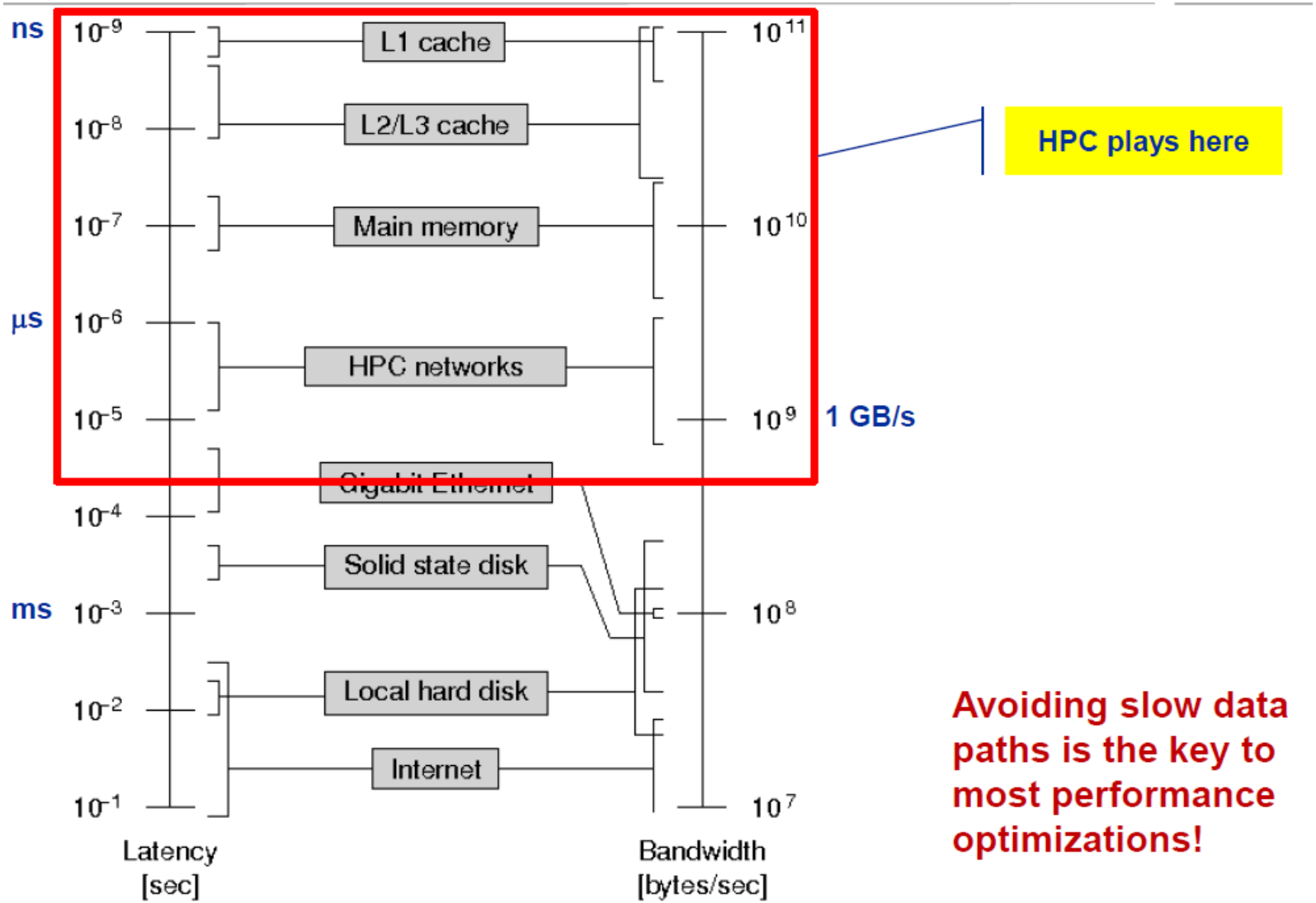


Figure: source: multicore tutorial (SC12) by Georg Hager and Gerhard Wellein

Hardware: memory hierarchy - cache

- How Locality affects scheduling algorithm selection: poor locality leads to long latency to fetch data from main memory ⇒ thread is blocked

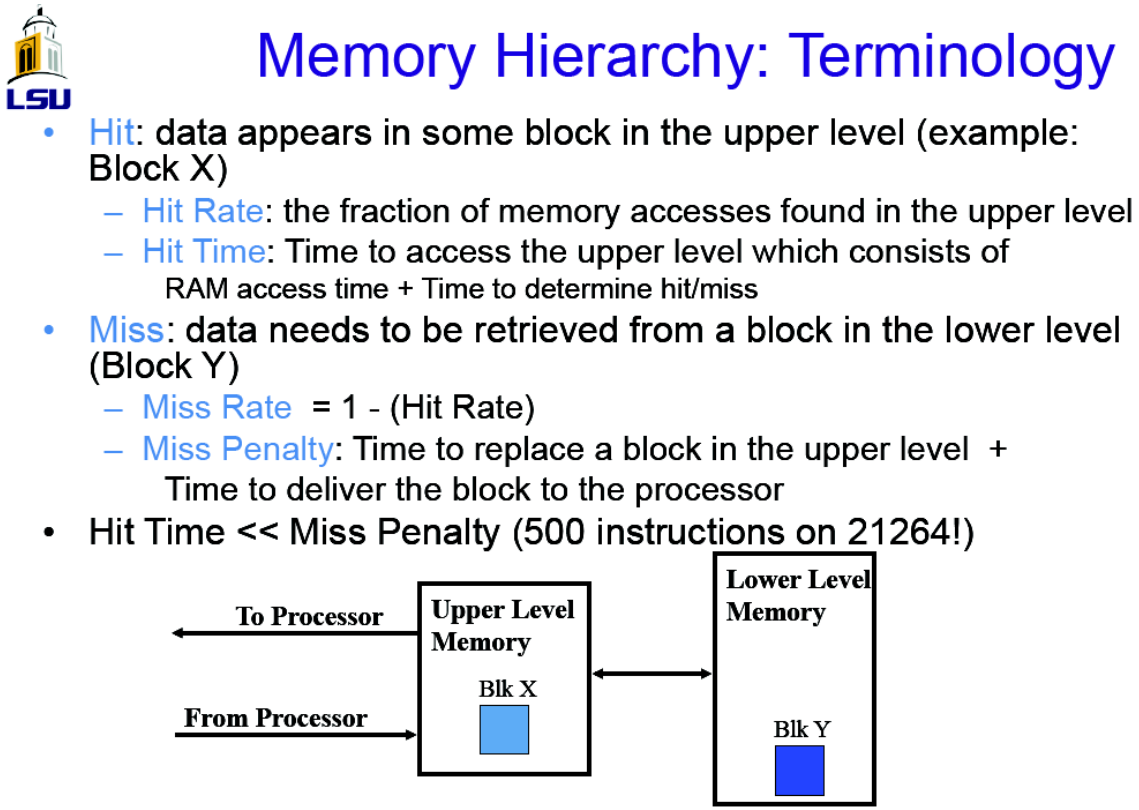
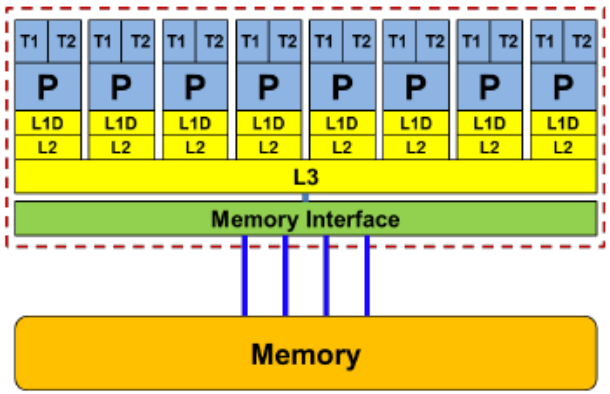


Figure: source: T. Sterling, Louisiana State University, SC12 Tutorial

Hardware: floating-point peak performance on multicore CPU



Intel Xeon
"Sandy Bridge EP" socket
4,6,8 core variants available

Floating Point (FP) Performance:

$P = n_{core} * F * S * \nu$

n_{core}	number of cores:	8
F	FP instructions per cycle: (1 MULT and 1 ADD)	2
S	FP ops / instruction: (256 Bit SIMD registers – "AVX")	4 (dp) / 8 (sp)
ν	Clock speed :	~2.7 GHz

TOP500 rank 1 (1995) → **P = 173 GF/s (dp)** / 346 GF/s (sp)

But: P=5 GF/s (dp) for serial, non-SIMD code

Figure: source: multicore tutorial (SC12) by Georg Hager and Gerhard Wellein

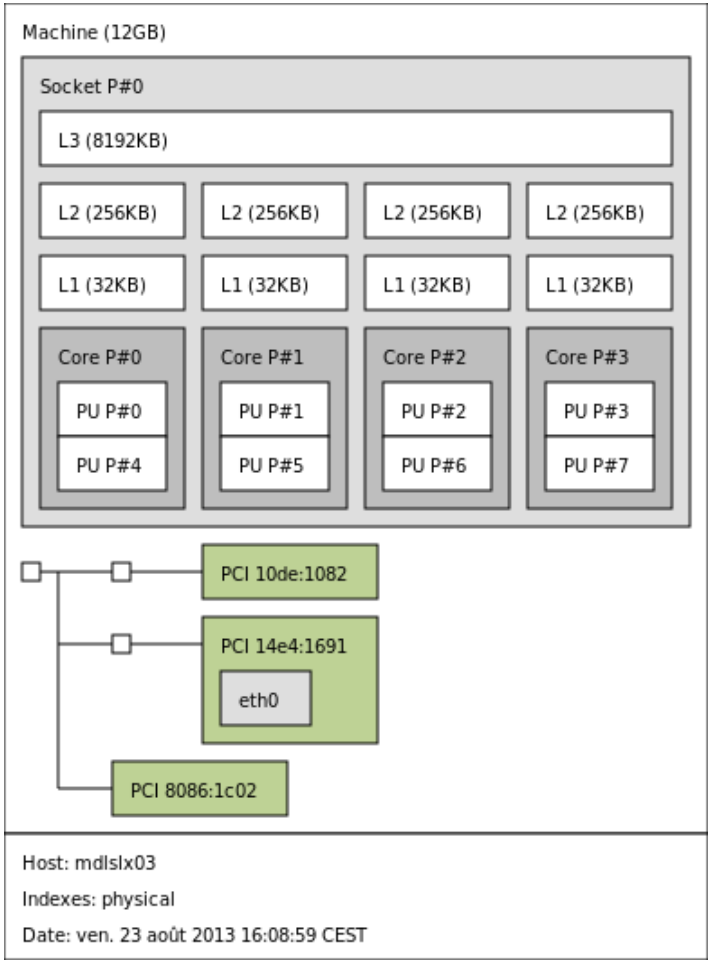
Hardware: linux tools to probe hardware features

- `cat /proc/cpuinfo | /bin/egrep 'processor|model name|cache size|core|sibling|physical'`
  - numérotation attribuée par le noyau linux
  - `core id`: numéro d'un cœur de CPU
  - `physical id`: numéro de socket
  - `siblings`: nombre de processing unit (PU) / *hardware thread* d'un CPU (socket)
  - pour un même `physical id`, si le nombre de cœur est égal à `siblings`, alors l'*hyperthreading* est déactivé
- `(sudo) lspci`
- `hwloc-ls / lstopo` (hwloc - hardware locality), outil utilisé par MPI pour le placement de tâche
- `lshw` et `lshw-gtk` (hardware LiSter)

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 42
model name      : Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
stepping        : 7
microcode       : 0x1a
cpu MHz         : 1600.000
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 0
cpu cores       : 4
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
:
```

Hardware: linux tools to probe hardware features

- `cat /proc/cpuinfo | /bin/egrep 'processor|model name|cache size|core|sibling|physical'`
  - numérotation attribuée par le noyau linux
  - `core id`: numéro d'un cœur de CPU
  - `physical id`: numéro de socket
  - `siblings`: nombre de processing unit (PU) / *hardware thread* d'un CPU (socket)
  - pour un même `physical id`, si le nombre de cœur est égal à `siblings`, alors l'*hyperthreading* est déactivé
- `(sudo) lspci`
- `hwloc-ls / lstopo` (hwloc - hardware locality), outil utilisé par MPI pour le placement de tâche
- `lshw` et `lshw-gtk` (hardware LiSter)
- TODO: lancer `hwloc-ls` sur `odette`. [2]



[2] Un article sur hwloc

Caractéristiques d'un code dit HPC

- Code qui est exécuté efficacement sur un super-calculateur:
  - $\Rightarrow$ bonnes propriétés de scalabilité faible et forte
- Des entrées-sorties performantes: utilisation de bibliothèques adaptées aux systèmes de fichier parallèle (Lustre, GPFS, ...); capacité à pouvoir reprendre un calcul interrompu
- Composabilité: mise en œuvre du parallélisme dans les bibliothèques de calcul: partage du communicateur MPI (e.g. éviter d'utiliser systématiquement `MPI_COMM_WORLD`)

Exemple de ALE3D / Lulesh https://codesign.llnl.gov/lulesh.php

MPI on multicore

- One MPI process per core
  - Each MPI process is a single thread
- One MPI process per node
  - MPI processes are multithreaded
  - One thread per core
  - aka Hybrid model

Parallel programming models

- **Definition:** the languages and libraries that create an abstract view of the machine (hide low-level details)
- **Control**
  - How is parallelism created?
  - How are dependencies enforced?
- **Data**
  - Shared or private?
  - How is shared data accessed or private data communicated?
- **Synchronization**
  - What operations can be used to coordinate parallelism
  - What are the atomic (indivisible) operations?

Slide derived from M. Zahran

Parallel programming models

- You can run any paradigm on any hardware (e.g. an MPI on shared - memory)
- The same program can have different type of parallel paradigms
- The hardware itself can be heterogeneous
- The whole challenge of parallel programming is to make the best use of the underlying hardware to exploit the different type of parallelisms

Slide derived from M. Zahran

Parallel programming models...

... on multicore multisocket nodes

- **Shared-memory (intra-node)**
  - Good old MPI (current standard: 2.2)
  - OpenMP (current standard: 3.0)
  - POSIX threads
  - Intel Threading Building Blocks (TBB)
  - Cilk++, OpenCL, StarSs,... you name it
- **Distributed-memory (inter-node)**
  - MPI (current standard: 2.2)
  - PVM (gone)
- **Hybrid**
  - Pure MPI
  - MPI+OpenMP
  - MPI + any shared-memory model
  - MPI (+OpenMP) + CUDA/OpenCL/...

All models require awareness of *topology* and *affinity* issues for getting best performance out of the machine!

Figure: source: multicore tutorial (SC12) by Georg Hager and Gerhard Wellein

Parallel programming models and Performance portability

- **Problem:**
  - Many architecture: x86_64, Power8, GPU, Xeon Phi, ...
  - Is it possble to write the application code **once**, and running efficiently (as best as possible) on each architecture listed above ?
- **Possible solutions:**
  - Use directives: OpenMP 4.0, OpenACC
  - Use active librairies: e.g. Kokkos [3], RAJA [4], ...
- A detail overview at `https://asc.llnl.gov/DOE-COE-Mtg-2016/`
- An up-to-date research article on the subject: `https://asc.llnl.gov/DOE-COE-Mtg-2016/talks/2-12_Martineau.p`

[3] A tutorial about kokkos: GTC2016 kokkos tutorial.
[4] RAJA overview article.

Supercomputing trends

It is not just "exaflops" – we are changing the whole computational model

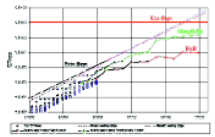
Current programming systems have WRONG optimization targets

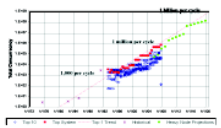
Old Constraints

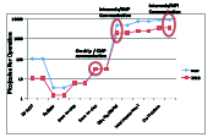
- **Peak clock frequency** *as primary limiter for performance improvement*
- **Cost:** *FLOPs* are biggest cost for system: *optimize for compute*
- **Concurrency**: Modest growth of parallelism by adding nodes
- **Memory scaling***: maintain byte per flop capacity and bandwidth*
- **Locality***: MPI+X model (uniform costs within node & between nodes)*
- **Uniformity**: *Assume uniform system performance*
- **Reliability***: It's the hardware's problem*

New Constraints

- **Power** *is primary design constraint for future HPC system design*
- **Cost:** *Data movement dominates: optimize to minimize data movement*
- **Concurrency:** *Exponential growth of parallelism within chips*
- **Memory Scaling:** *Compute growing 2x faster than capacity or bandwidth*
- **Locality***: must reason about data locality and possibly topology*
- **Heterogeneity**: *Architectural and performance non-uniformity increase*
- **Reliability***: Cannot count on hardware protection alone*







BERKELEY LAB

Fundamentally breaks our current programming paradigm and computing ecosystem

35

Figure: Horst Simon, LBNL

Ten things every programmer must know about hardware

- Data types
- Boolean algebra
- Caches - memory hierarchies
- Cache coherence
- Virtual Memory
- Pipelining
- Memory layout of data structures (arrays, linked lists, ...)
- Some assembly programming
- Basic compiler optimizations
- Memory bandwidth constratins

source: http://www.futurechips.org/tips-for-power-coders/programmer-hardware.html

Optimizing parallel programs - OpenMP example

http://www.futurechips.org/tips-for-power-coders/writing-optimizing-parallel-programs-complete.html

Sources of Performance Degradation

- **Latency:** Waiting for access to memory or other parts of the system
- **Overhead:** Extra work that has to be done to manage program concurrency and parallel resources the real work you want to perform
- **Starvation:** Not enough work to do due to insufficient parallelism or poor load balancing among distributed resources
- **Contention:** Delays due to fighting over what task gets to use a shared resource next. Network bandwidth is a major constraint.