

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 developer 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-free.md>



<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-performance-computing>
<http://www.admin-magazine.com/HPC/Articles/Singularity-A-Container-for-HPC>
- 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/>
<https://hub.docker.com/r/johntfoster/trilinos/>
 - autre exemple:
<https://trilinos.org/packages/web-trilinos/webtrilinos-docker>



Introduction

- **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

One woman can make a baby in 9 months.

Can 9 women make a baby in 1 month?

But 9 women can make 9 babies in 9 months.

Figure: source: [John Urbanic, Pittsburgh Supercomputing Center](#)



Parallel Computing: *devinette cuisine*

Dévinette #1

Combien de temps pour faire une tarte aux pommes

4×1 minutes



4×1 minutes



1×5 minutes



1×30 minutes



43 minutes tout seul

xx minutes à 2 ?



Parallel Computing: *devinette cuisine*

Dévinette #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 ?



Parallel Computing: *devinette cuisine*

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 économie ?



Parallel Computing: *devinette cuisine*

Devinette #4

Combien de temps pour faire une tarte aux pommes

4×1 minutes



4×1 minutes



1×5 minutes



1×30 minutes



35 minutes à 3

et si peu de temps pour tout manger...



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



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 !



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 !



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...



Parallel Computing

One woman can make a baby in 9 months.

Can 9 woman make a baby in 1 month?

But 9 women can make 9 babies in 9 months.

Figure: source: [John Urbanic, Pittsburgh Supercomputing Center](#)

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



Types de parallélisme

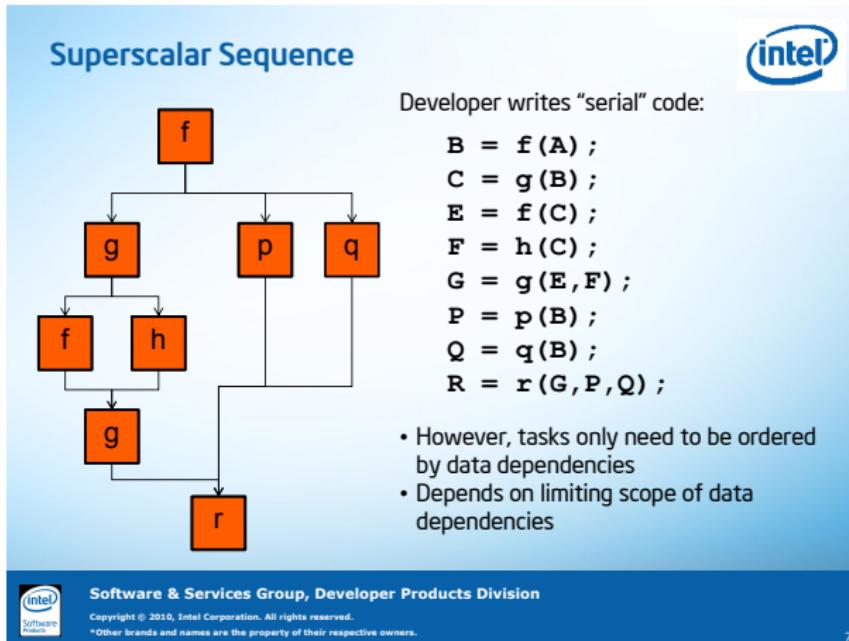
- Task parallelism
- Data parallelism
- Pipeline parallelism

<http://www.futurechips.org/parallel-programming-2/parallel-programming-clarifying-pipe>

- exemple : correction de copies d'examen



De la recette de cuisine à la notion de concurrence



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



Parallelism ≠ Concurrency

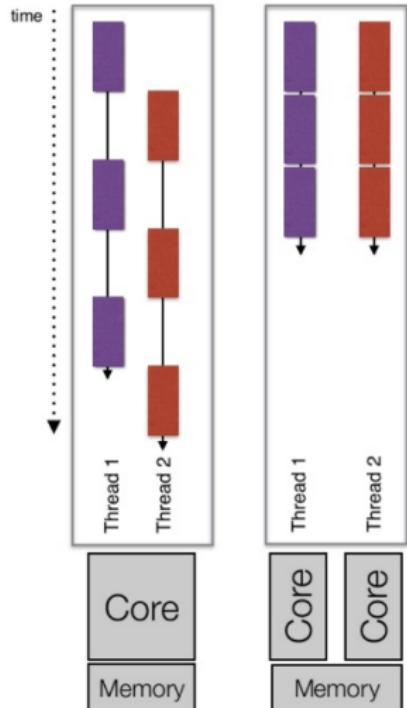
- **Concurrency:** At least two tasks are making progress at the same time frame.

- 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).



Parallelism ≠ Concurrency

- **Concurrency:** At least two tasks are making progress at the same time frame.

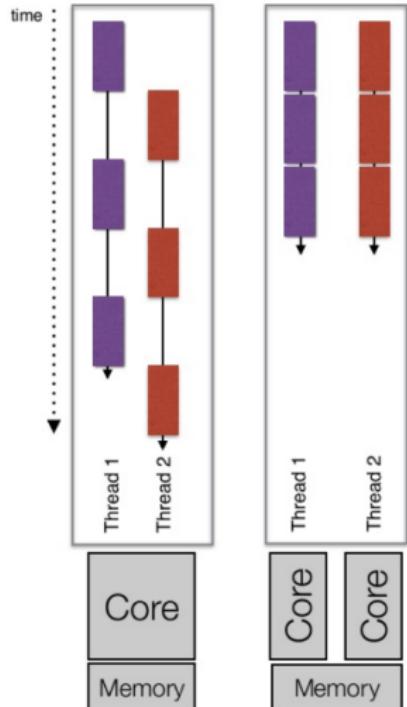
- Not necessarily at the same time
- Include techniques like time-slicing
- Can be implemented on a single processing unit
- Concept more general than parallelism
- Ex: multi-tasking on a single-core (time multiplexing)

- **Parallelism:** At least two tasks execute literally at the same time.

- Requires hardware with multiple processing units

- **Parallel computing** takes advantage of **concurrency** to:

- Solve large problems under bounded time
- Save on Wall clock time
- Overcome memory constraints
- Utilize non-local resources



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 ?



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



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

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



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/>

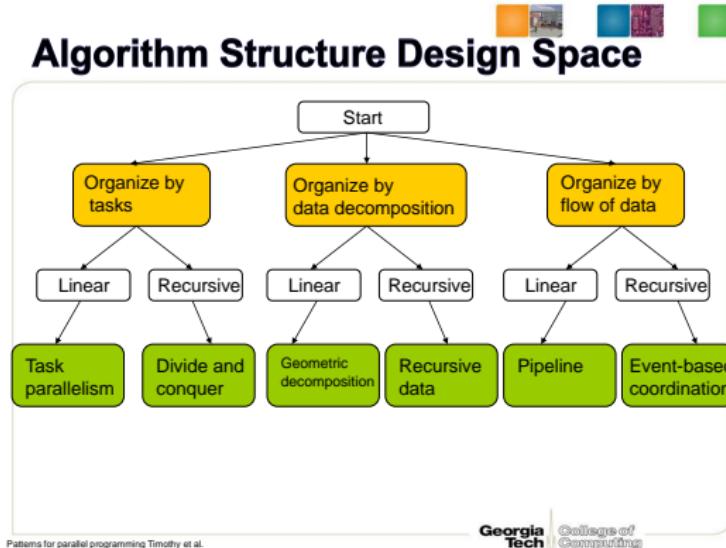


serial code = *data + algorithm + hardware*
parallel code =?

- Conception d'un code parallèle
- Coûts intrinsèques de la parallélisation
 - communication / échange de données:
 - latency: 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

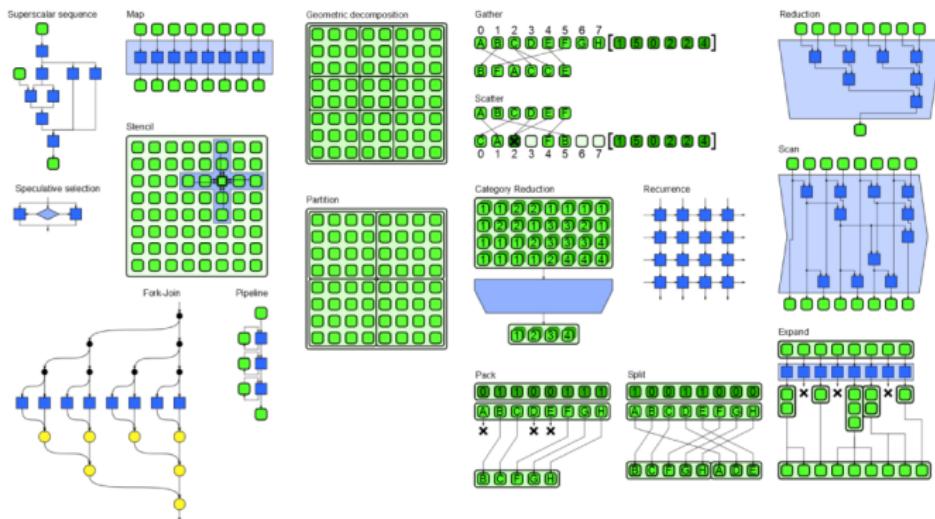
- **pattern : a basic structural entity of an algorithm**
- book
Structured Parallel Programming: Patterns for Efficient Computation
- implementation: Intel TBB, and many others
- OpenMP/OpenAcc for GPU/XeonPhi: pattern-based comparison:
map, stencil, reduce, scan, fork-join, superscalar sequence, parallel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

reference:

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



Parallel Patterns: Overview



reference: Structured Parallel Programming with Patterns, SC13 tutorial, by M. Hebenstreit, 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)

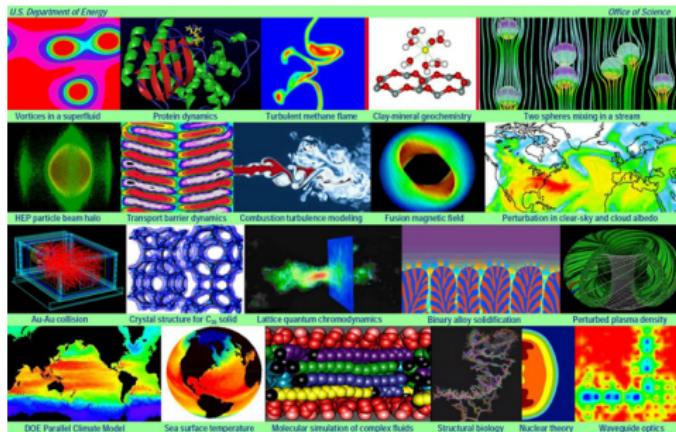


Figure: source: <http://www.scidac.gov/>

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](#)



Strategic importance of supercomputing:

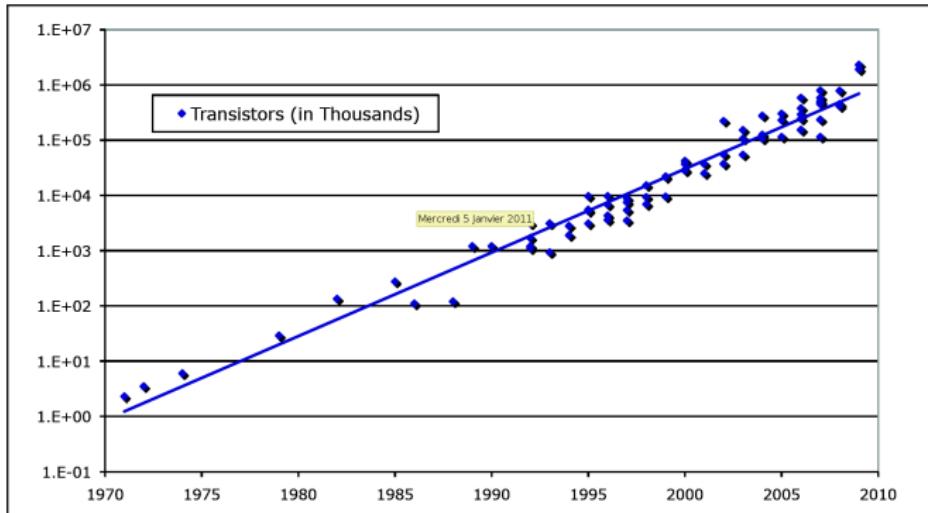
- essential of scientific discovery
- critical for national security
- fundamental contributor to the economy and competitiveness through use in engineering and manufacturing

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

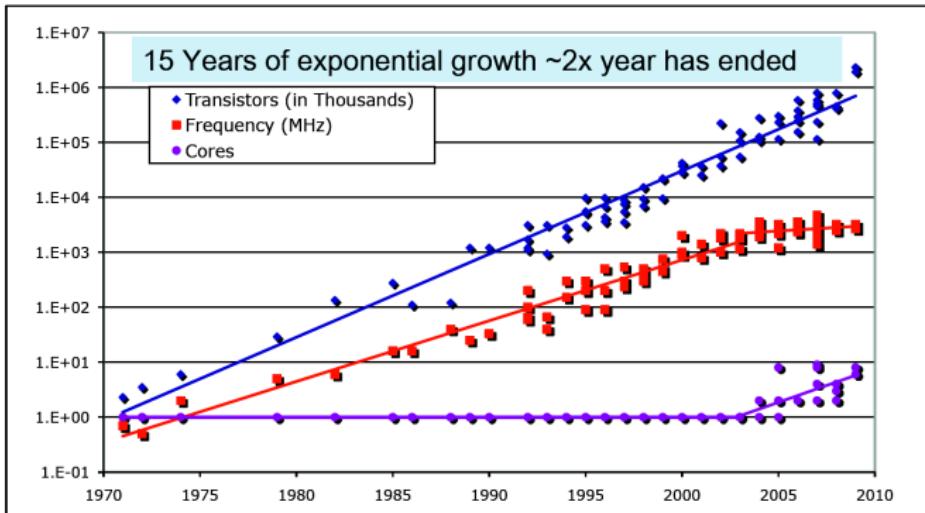


Data from Kunle Olukotun, Lance Hammond, Herb Sutter,
Burton Smith, Chris Batten, and Krste Asanović



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 Asanović



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.

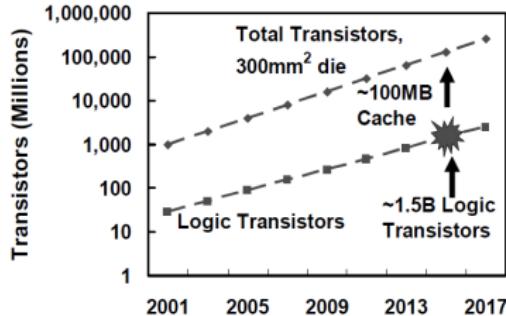


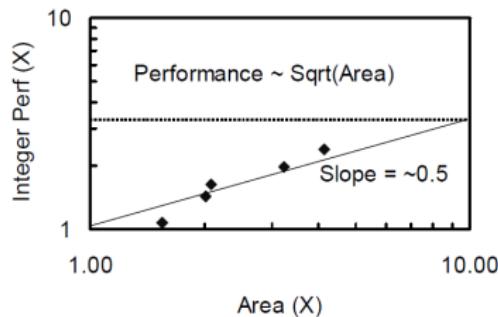
Figure 1: Transistor integration capacity



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

Example: Dual core with voltage scaling

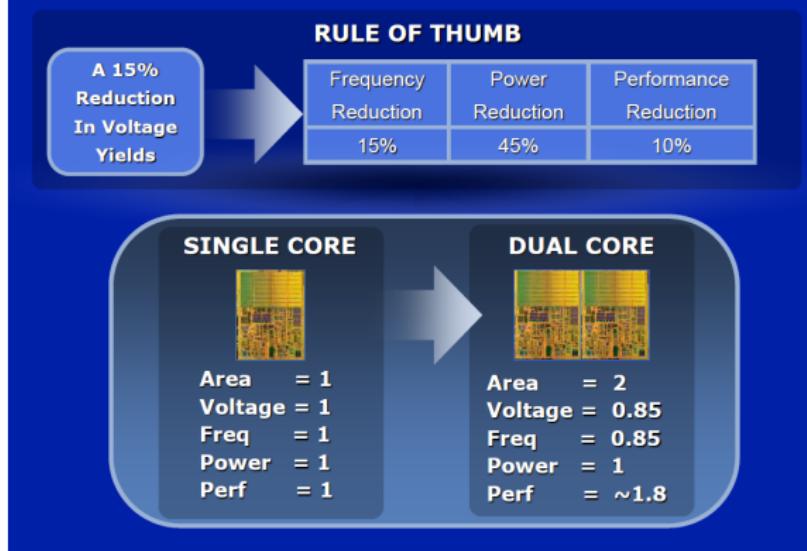


Figure: source: John Urbanic, Pittsburgh Supercomputing Center





Power Cost of Frequency

- Power \propto Voltage² x Frequency (V²F)
- Frequency \propto Voltage
- Power \propto Frequency³

	Cores	V	Freq	Perf	Power	PE (Bops/watt)
Superscalar	1	1	1	1	1	1
"New" Superscalar	1X	1.5X	1.5X	1.5X	3.3X	0.45X
Multicore	2X	0.75X	0.75X	1.5X	0.8X	1.88X

(Bigger # is better)

50% more performance with 20% less power

Preferable to use multiple slower devices, than one superfast device

63

Figure: source: [Scientific Computing for engineers, CS594, J. Dongarra](#)

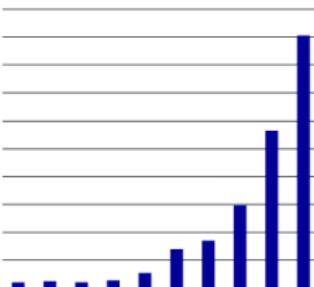


Moore's law - Towards multi-core architectures

 Moore's Law Reinterpreted

- Number of cores per chip doubles every 2 year, while clock speed decreases (not increases).
 - Need to deal with systems with millions of concurrent threads
 - Future generation will have billions of threads!
 - Need to be able to easily replace inter-chip parallelism with intra-chip parallelism
- Number of threads of execution doubles every 2 year

Average Number of Cores Per Supercomputer



Year	Average Number of Cores
1990	~10
1992	~20
1994	~40
1996	~80
1998	~160
2000	~320
2002	~640
2004	~1280
2006	~2560
2008	~5120
2010	~10240
2012	~20480
2014	~40960
2016	~81920
2018	~163840
2020	~327680
2022	~655360
2024	~1310720
2026	~2621440
2028	~5242880
2030	~10485760
2032	~20971520
2034	~41943040
2036	~83886080
2038	~167772160
2040	~335544320
2042	~671088640
2044	~1342177280
2046	~2684354560
2048	~5368709120
2050	~10737418240
2052	~21474836480
2054	~42949672960
2056	~85899345920
2058	~171798691840
2060	~343597383680
2062	~687194767360
2064	~1374389534720
2066	~2748779069440
2068	~5497558138880
2070	~10995116277760
2072	~21990232555520
2074	~43980465111040
2076	~87960930222080
2078	~175921860444160
2080	~351843720888320
2082	~703687441776640
2084	~1407374883553280
2086	~2814749767106560
2088	~5629499534213120
2090	~11258999068426240
2092	~22517998136852480
2094	~45035996273704960
2096	~90071992547409920
2098	~180143985094819840
2100	~360287970189639680
2102	~720575940379279360
2104	~1441151880758598720
2106	~2882303761517197440
2108	~5764607523034394880
2110	~11529215046068789760
2112	~23058430092137579520
2114	~46116860184275159040
2116	~92233720368550318080
2118	~184467440737100636160
2120	~368934881474201272320
2122	~737869762948402544640
2124	~1475739525896805089280
2126	~2951479051793610178560
2128	~5902958103587220357120
2130	~11805916207174440714240
2132	~23611832414348881428480
2134	~47223664828697762856960
2136	~94447329657395525713920
2138	~188894659314791051427840
2140	~377789318629582102855680
2142	~755578637259164205711360
2144	~1511157274518328411422720
2146	~3022314549036656822845440
2148	~6044629098073313645690880
2150	~12089258196146627291381760
2152	~24178516392293254582763520
2154	~48357032784586509165527040
2156	~96714065569173018331054080
2158	~193428131138346036662108160
2160	~386856262276692073324216320
2162	~773712524553384146648432640
2164	~1547425049106768293296865280
2166	~3094850098213536586593730560
2168	~6189700196427073173187461120
2170	~1237940039285414634637482240
2172	~2475880078570829269274964480
2174	~4951760157141658538549928960
2176	~9903520314283317077099857920
2178	~19807040628566634154199715840
2180	~39614081257133268308399431680
2182	~79228162514266536616798863360
2184	~158456325288533073233597726720
2186	~316912650577066146467195453440
2188	~633825301154132292934390906880
2190	~1267650602308264585868781813760
2192	~2535301204616529171737563627520
2194	~5070602409233058343475127255040
2196	~10141204818466116686950254510080
2198	~20282409636932233373900509020160
2200	~40564819273864466747801018040320
2202	~81129638547728933495602036080640
2204	~162259277095457866991204072161280
2206	~324518554190915733982408144322560
2208	~649037108381831467964816288645120
2210	~1298074216763662935929632577290240
2212	~2596148433527325871859265154580480
2214	~5192296867054651743718530309160960
2216	~10384593734109303487437066018321920
2218	~20769187468218606974874132036643840
2220	~41538374936437213949748264073287680
2222	~83076749872874427899496528146575360
2224	~166153499745748855798993056293150720
2226	~332306999491497711597986112586301440
2228	~664613998982995423195972225172602880
2230	~1329227997965990846391944450345257600
2232	~2658455995931981692783888900687515200
2234	~5316911991863963385567777801375030400
2236	~10633823983727866771135555602750608000
2238	~21267647967455733542271111205501216000
2240	~42535295934911467084542222411002432000
2242	~85070591869822934169084444822004864000
2244	~170141183739645868338168889644009728000
2246	~340282367479291736676337779288019456000
2248	~680564734958583473352675558576038912000
2250	~1361129469917166946705351117152078240000
2252	~2722258939834333893410702234304156480000
2254	~5444517879668667786821404468608312960000
2256	~10889035759337335573642808937216625920000
2258	~21778071518674671147285617874433251840000
2260	~43556143037349342294571235748866503680000
2262	~87112286074698684589142471497733007360000
2264	~174224572149397369178284942995466014720000
2266	~348449144298794738356569885990932029440000
2268	~696898288597589476713139771981864058880000
2270	~139379657719517895342627554396372811760000
2272	~278759315438535790685255108792745623520000
2274	~557518630877071581370510217585491246400000
2276	~111503726175414316274102043517098249200000
2278	~223007452350828632548204087034196498000000
2280	~446014904701657265096408174068392996000000
2282	~892029809403314520192816348136785992000000
2284	~1784059618806629040385632696273771984000000
2286	~3568119237613258080771265392547543968000000
2288	~7136238475226516161542530785095087936000000
2290	~14272476950453032323085061570190175872000000
2292	~28544953900856064646170123140380351744000000
2294	~57089907801712129292340246280760703488000000
2296	~114179815603424258584680492561521407376000000
2298	~228359631206848517169360985123042814752000000
2300	~456719262413697034338721970246085629040000000
2302	~913438524827394068677443940492171258080000000
2304	~1826877049654788137354887880984342516160000000
2306	~3653754099309576274709775761968685032320000000
2308	~7307508198619152549419551523937370064640000000
2310	~14615016397238305098839103047874740129280000000
2312	~29230032794476610197678206095749480258560000000
2314	~58460065588953220395356412191498960517120000000
2316	~116920131177866406890712824382997921034240000000
2318	~233840262355732813781425648765995842068480000000
2320	~467680524711465627562851297531991684136960000000
2322	~935361049422931255125702595063983368273920000000
2324	~1870722098845862510251405190127966736557840000000
2326	~3741444197691725020502810380255933473115680000000
2328	~7482888395383450041005620760511866946231360000000
2330	~1496577679076690008201124152102373389262720000000
2332	~2993155358153380016402248304204746778525440000000
2334	~5986310716306760032804496608409493557050880000000
2336	~11972621432613520065608993216818987114101760000000
2338	~23945242865227040131217986433637974228203520000000
2340	~47890485730454080262435972867275948456407040000000
2342	~95780971460908160524871945734551896912814080000000
2344	~191561942921816321049743891469103793825628160000000
2346	~383123885843632642099487782938207587651256320000000
2348	~766247771687265284198975565876415175302512640000000
2350	~1532495543374530568397951131752830350655025280000000
2352	~3064991086749061136795902263505660701300050560000000
2354	~6129982173498122273591804527011321402600101120000000
2356	~12259964346996244547183609054022642805200202240000000
2358	~24519928693992489094367218108045285610400404480000000
2360	~49039857387984978188734436216090571220800808960000000
2362	~98079714775969956377468872432181142441601617920000000
2364	~196159429551939912754937744864362284883203235840000000
2366	~392318859103879825509875489728724569766406471680000000
2368	~784637718207759651019750979457449139532812943360000000
2370	~156927543641551930203950195891489827865645886720000000
2372	~313855087283103860407900391782978555731291773440000000
2374	~627710174566207720815800783565957111462583546880000000
2376	~125542034913241544163160156713191422292516709360000000
2378	~251084069826483088326320313426382844585033418720000000
2380	~502168139652966176652640626852765689170066837440000000
2382	~1004336279305932353305281253705531378340133674880000000
2384	~200867255861186470661056250741106275668026734960000000
2386	~401734511722372941322112501482212551336053469920000000
2388	~803469023444745882644225002964425102672106939840000000
2390	~160693804688949176528845000592885020534421387968000000
2392	~321387609377898353057690001185770041068842775936000000
2394	~642775218755796706115380002371540082137685551872000000
2396	~1285550437511593412230760004743080164275371103744000000
2398	~2571100875023186824461520009486160328550742207488000000
2400	~5142201750046373648923040018972320657101484414960000000
2402	~10284403500092747297846080379446413142029688829920000000
2404	~20568807000185494595692160758892826284059377659840000000
2406	~41137614000370989191384320157785652568118755319680000000
2408	~82275228000741978382768640315571305136237510639360000000
2410	~16455045600148395665553720631142610272475022127840000000
2412	~32910091200296791331107441262285220544950044255680000000
2414	~65820182400593582662214882524570441089800088511360000000
2416	~131640364801187165324429765049140882178400177022720000000
2418	~263280729602374330648859530098281764356800354045440000000
2420	~526561459204748661297719060196563528713600708090880000000
2422	~1053122918409497322595438120393127057427201416181760000000
2424	~2106245836818994645188876240786254114854402832363520000000
2426	~4212491673637989290377752481572508229708805664727040000000
2428	~8424983347275978580755504963145016559417611329544080000000
2430	~16849966694551957161511009926290232118835226658880160000000
2432	~33699933389103914323022009852580464237670453317760320000000
2434	~67399866778207828646044009705160928475340906635520640000000
2436	~134799733556415657292088009410321856950681813270641280000000
2438	~269599467112831314584176009220643713901363626541282560000000
2440	~539198934225662629168352009441287427802727253082565120000000
2442	~1078397868451325258336704009882574855605454506165130240000000
2444	~2156795736902650516673408009765149711210909012325260480000000
2446	~4313591473805301033346816009530299422421818024650520960000000
2448	~862718294761060206669363200946059884484363604

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

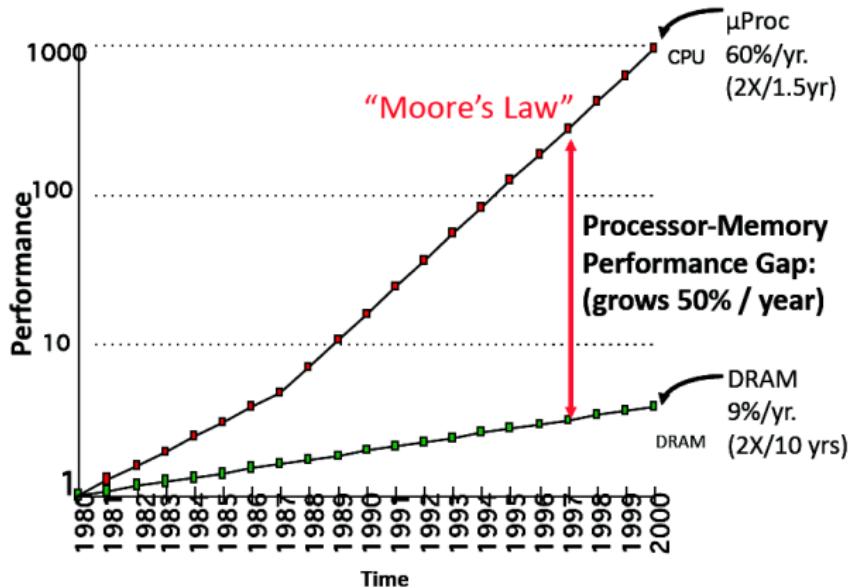


Figure: source: [T. Sterling, Louisiana State University](#)



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

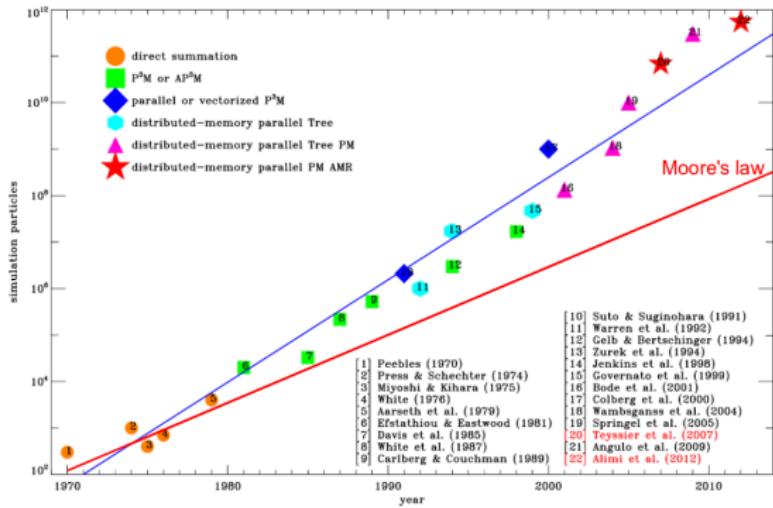
OSTP Briefing, 4 May 2004

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



Progrès algorithmiques

Cosmological N body simulations



source: R. Teyssier

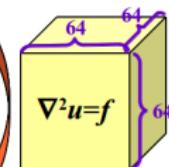


Progrès algorithmiques

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^5



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

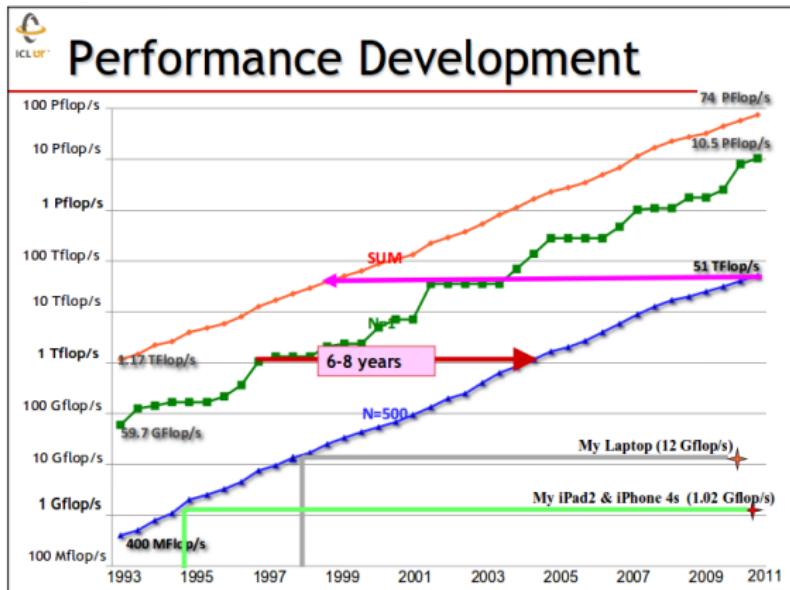
*Six-months is reduced to 1 s

OSTP Briefing, 4 May 2004

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



Evolution des performances brutes



“Automatic” Parallelism in Modern Machines

- ◆ Bit level parallelism
 - within floating point operations, etc.
- ◆ Instruction level parallelism (ILP)
 - multiple instructions execute per clock cycle
- ◆ Memory system parallelism
 - overlap of memory operations with computation
- ◆ OS parallelism
 - multiple jobs run in parallel on commodity SMPs

Limits to all of these -- for very high performance, need user to identify, schedule and coordinate parallel tasks

83

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

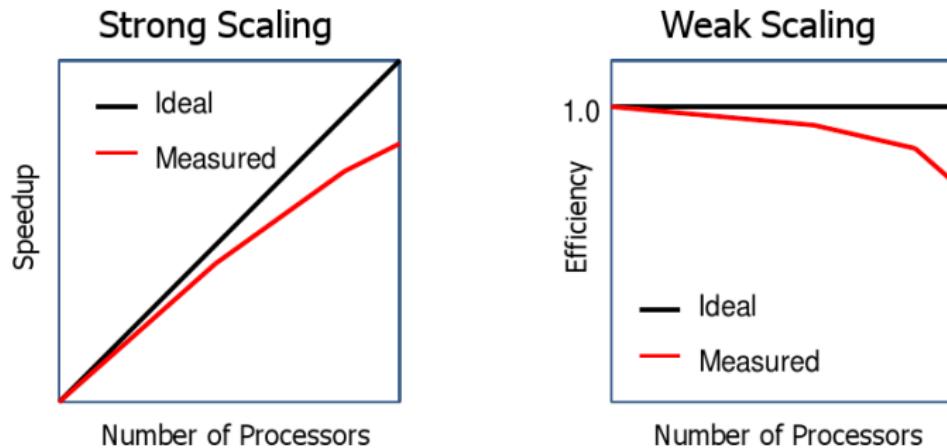
- **scalable efficiency:** Ratio of the actual performance to the best possible performance.

$$\text{Efficiency} = \frac{\text{execution_time_on_1_processor}}{\text{execution_time_on_N_processor} \times N}$$

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.



Supercomputers

Qu'est ce qu'un super-calculateur ?

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



Supercomputers

Qu'est ce qu'un super-calculateur ?

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



Supercomputers

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/address, 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)

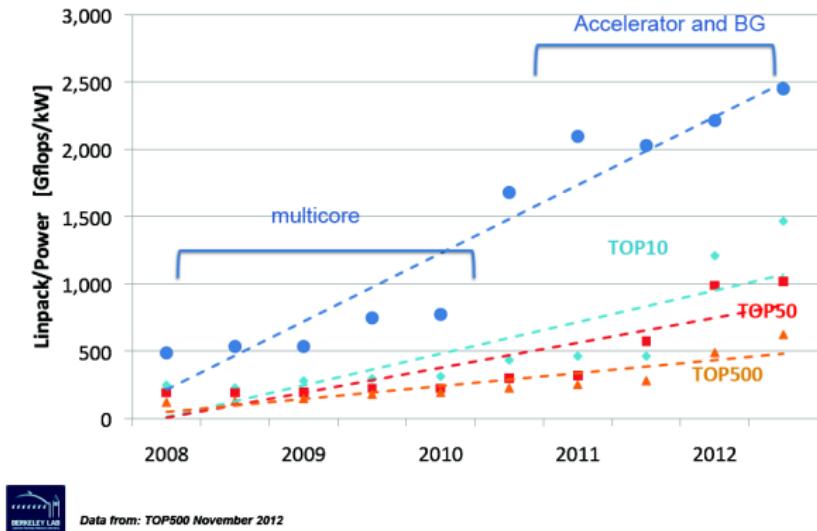
- **Multicore:** Maintain complex cores, and replicate (x86, SPARC, Power7) (#3, 6, and 10)
- **Manycore/Embedded:** Use many simpler, low power cores from embedded (IBM BlueGene) (#2, 4, 5 and 9)
- **GPU/MIC/Accelerator:** Use highly specialized processors from gaming/graphics market space (NVidia Fermi,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



Supercomputers

Qu'est ce qu'un super-calculateur ?

Power Efficiency over Time



26

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](#) (*filesystem / 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)



Supercomputers / Mesure de performance

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

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

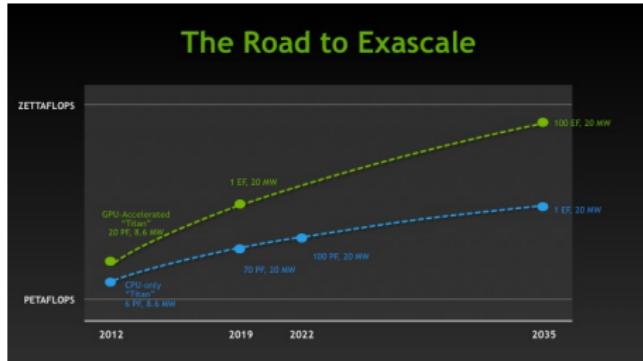


- Voir le tableau récapitulatif sur le site de [LLNL](#)
- [MPIBlib](#): P2P, collective MPI communication benchmark



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](https://doi.org/10.1177/1094342010391989))
 - Make a thorough assessment of needs, issues and strategies,
 - Develop a coordinated software roadmap,
 - 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 useful 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,...**)

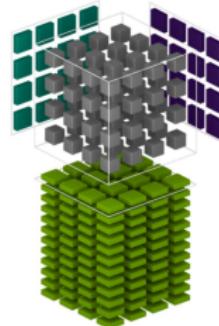


Exascale race: technological challenge(s)

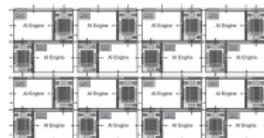
...also about building an economic ecosystem

- 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)



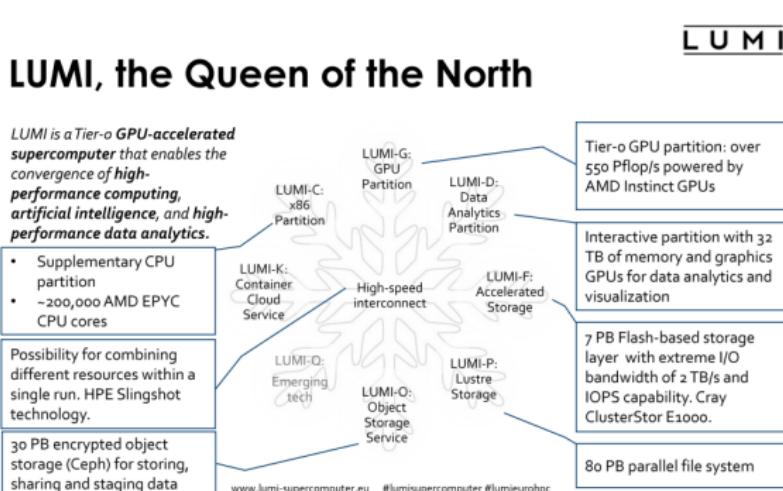
Pre-Exascale machines - architecture diversity !

- **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 processor 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

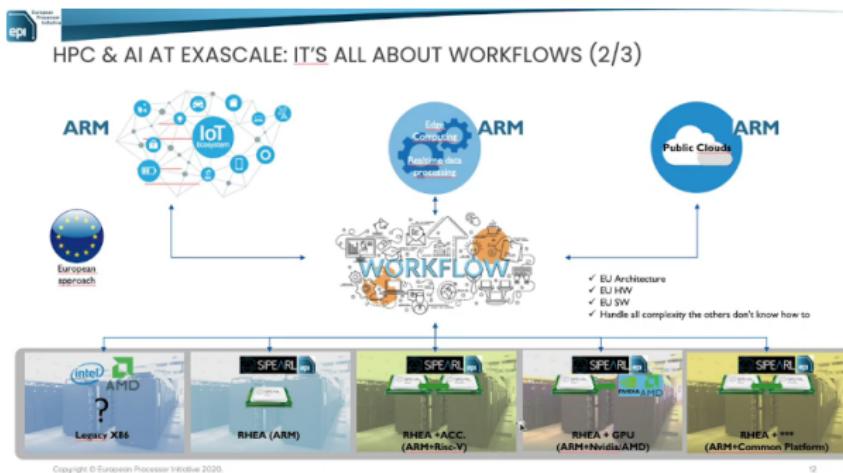


<https://www.european-processor-initiative.eu/project/epi/>



European Processor Initiative (EPI)

- project coordinated by ATOS/Bull:
 - hardware : an ARM-based processor 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

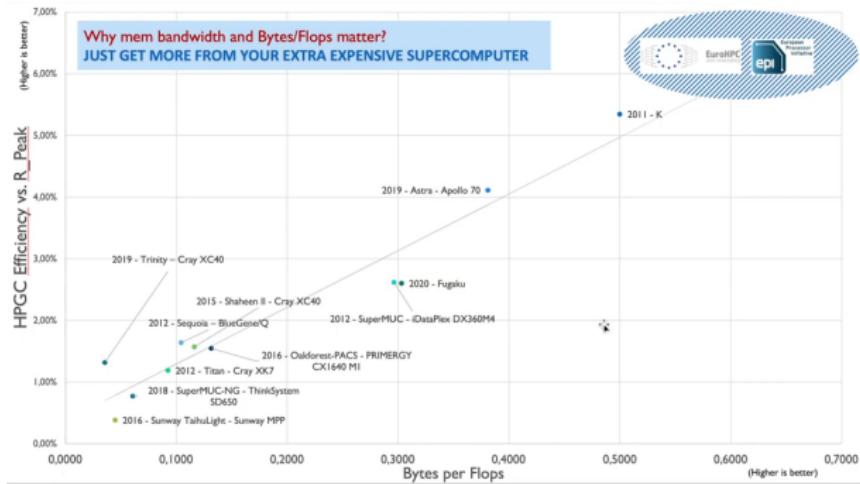


<https://www.european-processor-initiative.eu/project/epi/>



European Processor Initiative (EPI)

- project coordinated by ATOS/Bull:
 - hardware : an ARM-based processor 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/>



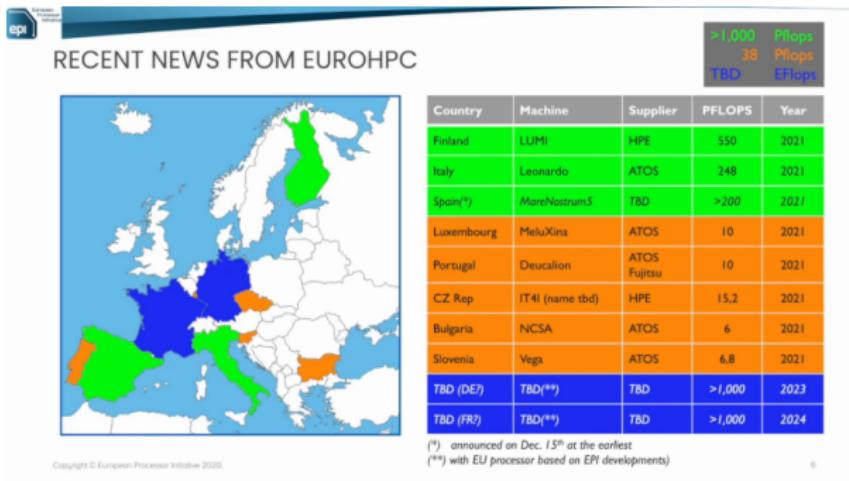
<https://eurohpc-ju.europa.eu/>, created in 2018; drives how and where supercomputing resources should be settled in Europe.



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

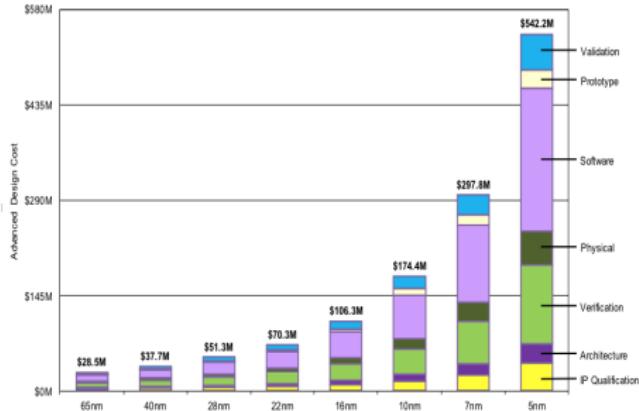
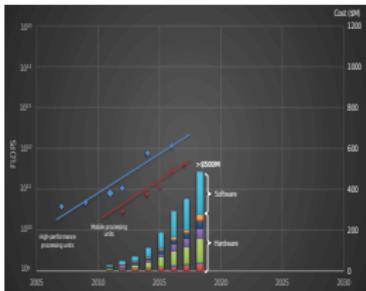


<https://eurohpc-ju.europa.eu/>, created in 2018; drives how and where supercomputing resources should be settled in Europe.



Beyond Moore's Law ...

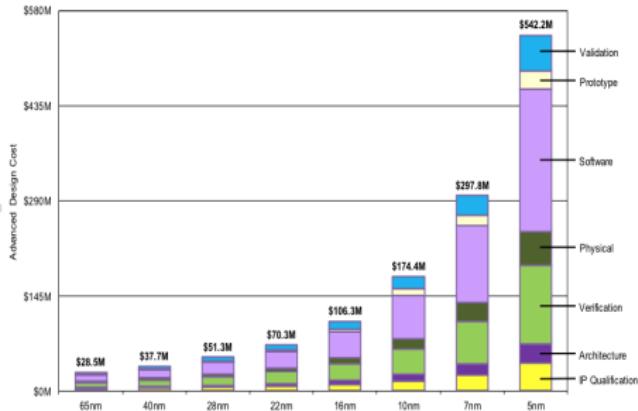
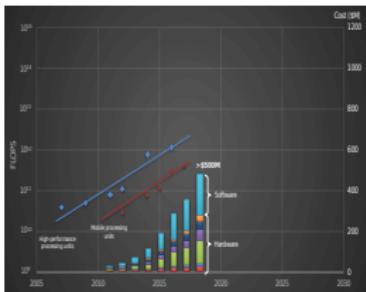
- Semiconductor sector still grows faster than GDP (PIB)
- Current chip makers (Intel, NVidia, Xilinx, ...) increasingly focus on specific applications : AI,... not necessarily HPC driven anymore
- **Skyrocketing cost of chip development limits hardware innovation.**
- June 2017 : **DARPA Electronics 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.enterprisotech.com/2018/07/24/darpa-effort-pushes-beyond-moores-law>

Beyond Moore's Law ...

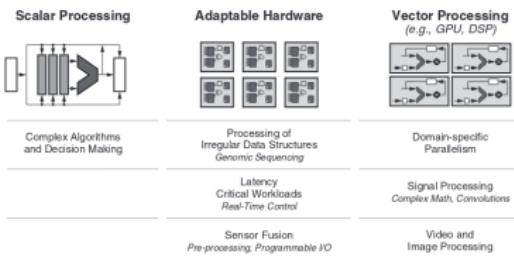
- **The cost of chip making... needs huge revenue to justify these expenses**
- Nvidia Volta/Turing (12 nm), next-gen in 7 nm
- AMD Vega / ZEN2 (2019): 7nm
- Xilinx Versal FPGA (2019): 7nm
- Fujitsu ARM 64 bit for HPC (2019-2020): 7nm
- **nothing beyond 3nm (if reachable) ? ⇒ 3D integration package**



<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**



WP901_02_002718



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\$**

Pre-Exascale Systems				Exascale Systems	
2013	2016	2018	2020	2021-2022	
 Mira Argonne IBM BG/Q Open	 Thor Argonne Intel/Cray KNL Open	 Summit ORNL IBM/Nvidia P9/Volta Open	 NERSC-9 LBNL TBD Open	 A21 Argonne Intel/Cray TBD Open	
 Titan ORNL Cray/Nvidia K20 Open	 CORI LBNL Cray/Intel Xeon/KNL Unclassified			 Frontier ORNL TBD Open	
 Sequoia LLNL IBM BG/Q Secure	 Trinity LANL/SNL Cray/Intel Xeon/KNL Secure	 Sierra LLNL IBM/Nvidia P9/Volta Secure	 Crossroads LANL/SNL TBD Secure	 El Capitan LLNL TBD Secure	

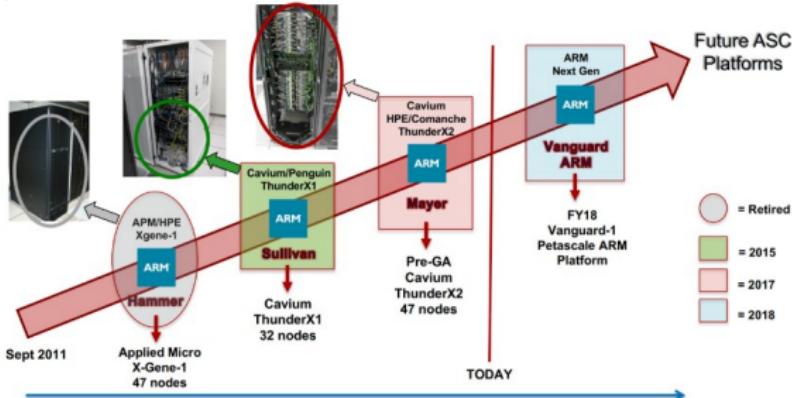
<https://www.nextplatform.com/2018/04/09/bidders-off-and-running-after-1-8-billion-doe-exascale-supercomputer-deals/>

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

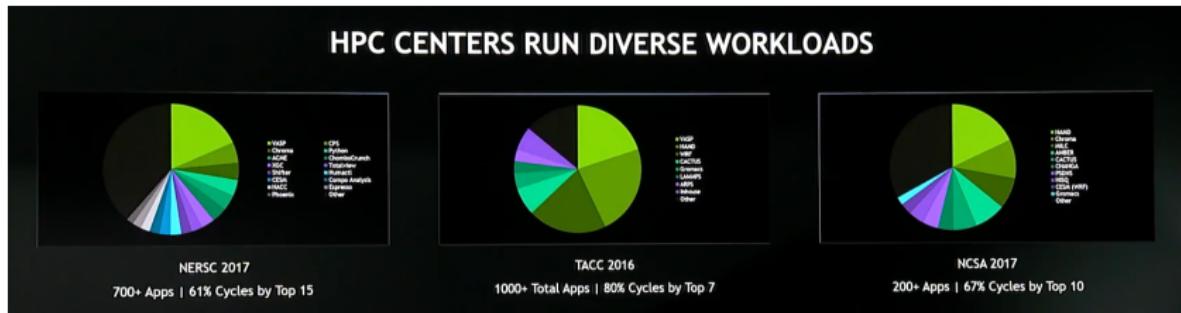


Multicore architectures: X86, ARM, ...

- **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

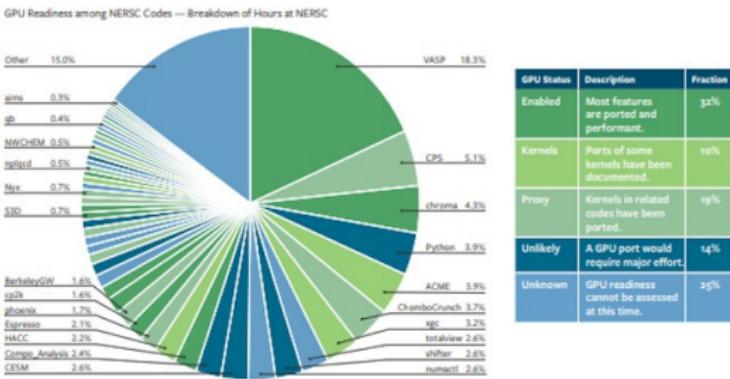


SC18: NVIDIA CEO talk



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

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



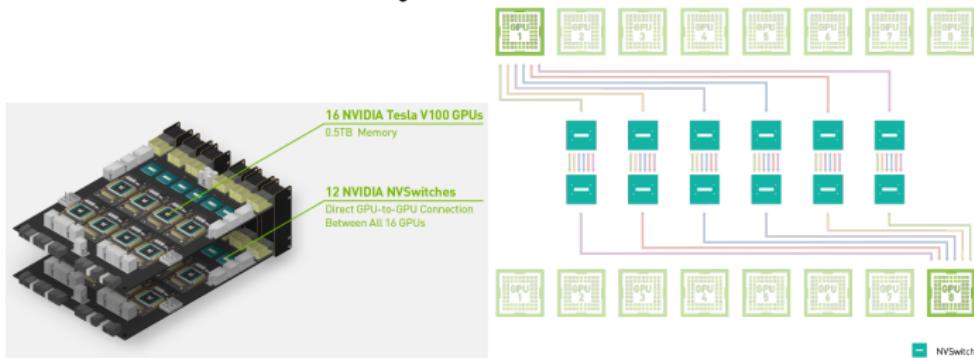
New fast interconnect by CRAY: named SHASTA, enabling new applications, real-time image processing data from telescopes (e.g. already existing ASTRON for Fast Radio Burst, ...)

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



Nvidia roadmap for HPC and AI - hardware

HGX-2 - heavy fat GPU-based node

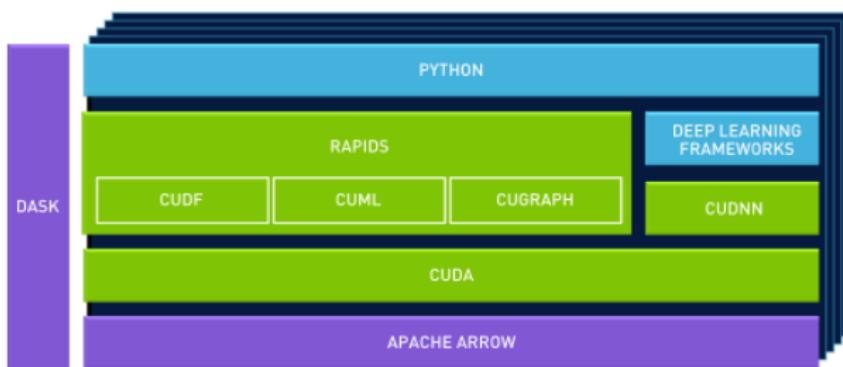


- Mostly for AI applications in the cloud, expensive (400 k\$)
- If Summit (#1 @ top500, 27 000 GPU) was assembled with HGX2 => 675 M\$ (for the GPUs only !)

Nvidia roadmap for HPC and AI - software

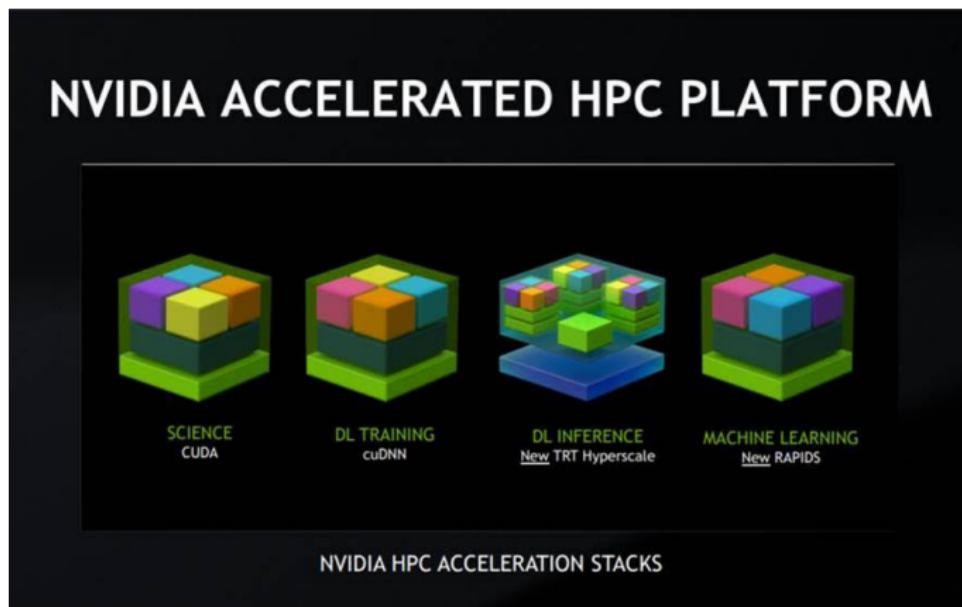
Semantic shift: HPC = simulation + AI

Machine Learning to Deep Learning: All on GPU



Nvidia roadmap for HPC and AI - software

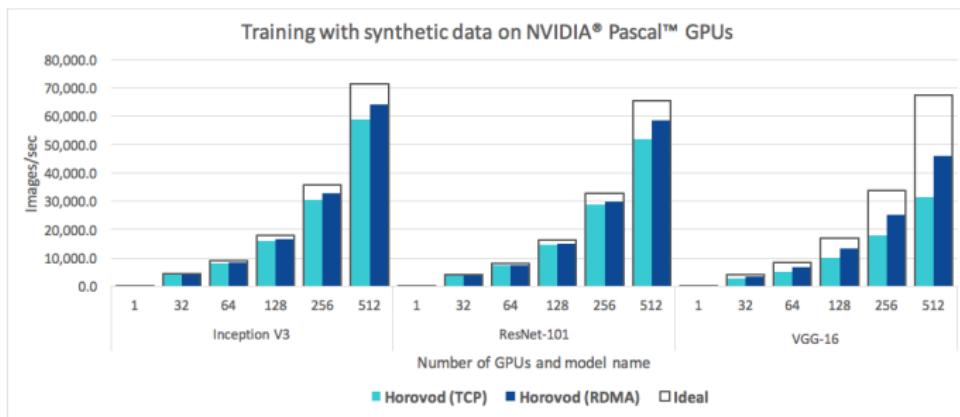
Semantic shift: HPC = simulation + AI



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 instantiates 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 survey¹
 - 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)

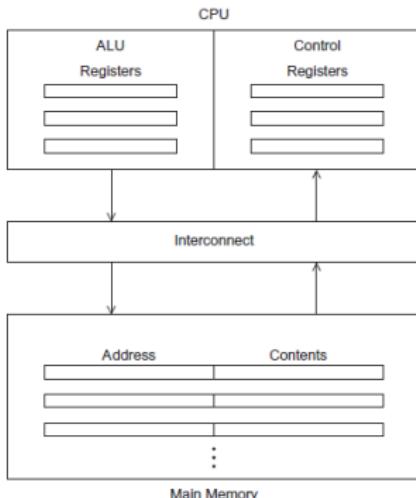
	Supercalculateurs	Valorisation moyennée en cts d'une heure/œur/machine	Valorisation en euros pour 1 Mh
CINES	Bull – Occigen	3,2	31 771 €
IDRIS	IBM – Nœuds larges Ada	2,2	21 761 €
	IBM BG/Q – Turing	0,7	7 254 €
TGCC	Bull – Irene SKL	4,3	43 523 €
	Bull – Irene KNL	1,3	12 903 €

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



Serial hardware

Von Neumann architecture



Multi-core CPU

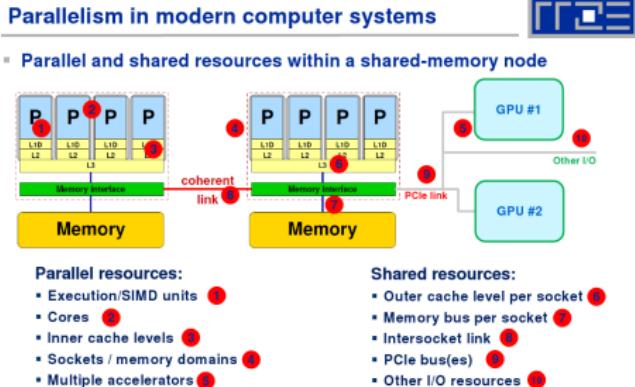


Figure: source: multicore tutorial (SC12) by Georg Hager and Gerhard

Figure: Peter Pacheco, SanFrancisco U. Wellein



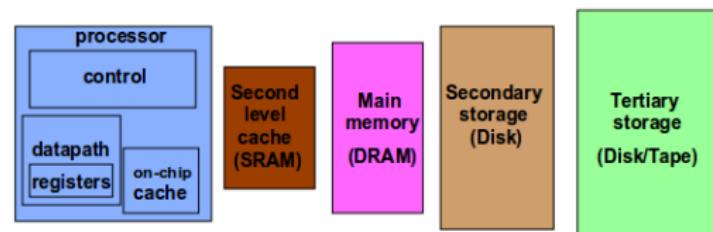
Processor Core Micro architecture

- 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 \Rightarrow need for intermediate staging area: **cache memory**
- Memory hierarchy tries to exploit locality



Speed	1ns	10ns	100ns	10ms	10sec
Size	B	KB	MB	GB	TB

Figure: source: [Scientific Computing for engineers, CS594, J. Dongarra](#)



Hardware: memory hierarchy - low / high latency



Capacity
Access Time
Cost

CPU Registers
100s Bytes
< 0.5 ns (typically 1 CPU cycle)

Cache
L1 cache:
10s-100s K Bytes
1-5 ns
\$10 / Mbyte

Main Memory
Few G Bytes
50ns- 150ns
\$0.02 / MByte

Disk
100s-1000s G Bytes
500000ns- 1500000ns
\$ 0.25 / GByte

Tape
infinite
sec-min
\$0.0014 / MByte

Levels of the Memory Hierarchy

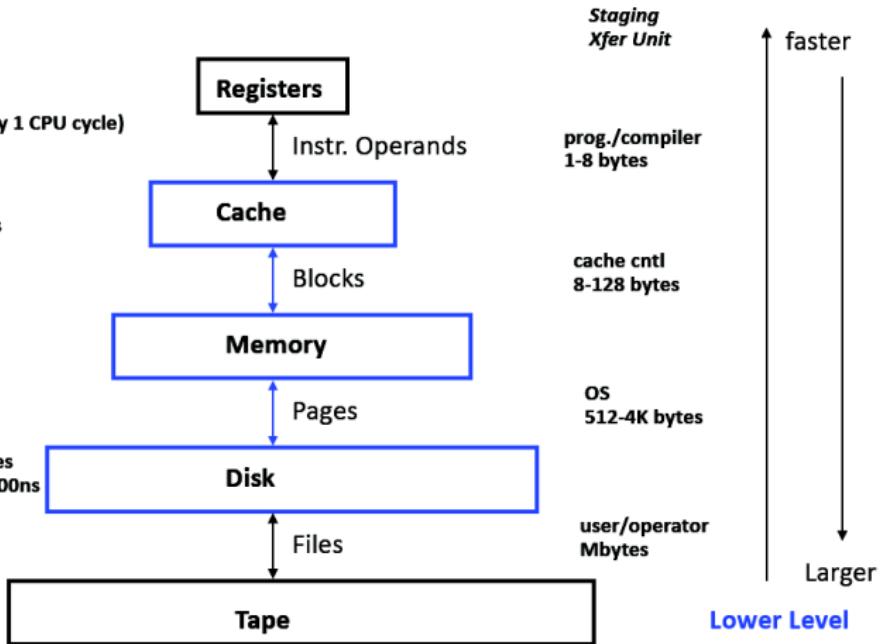


Figure: source: [T. Sterling, Louisiana State University, SC12 Tutorial](#)



Hardware: memory hierarchy - low / high latency

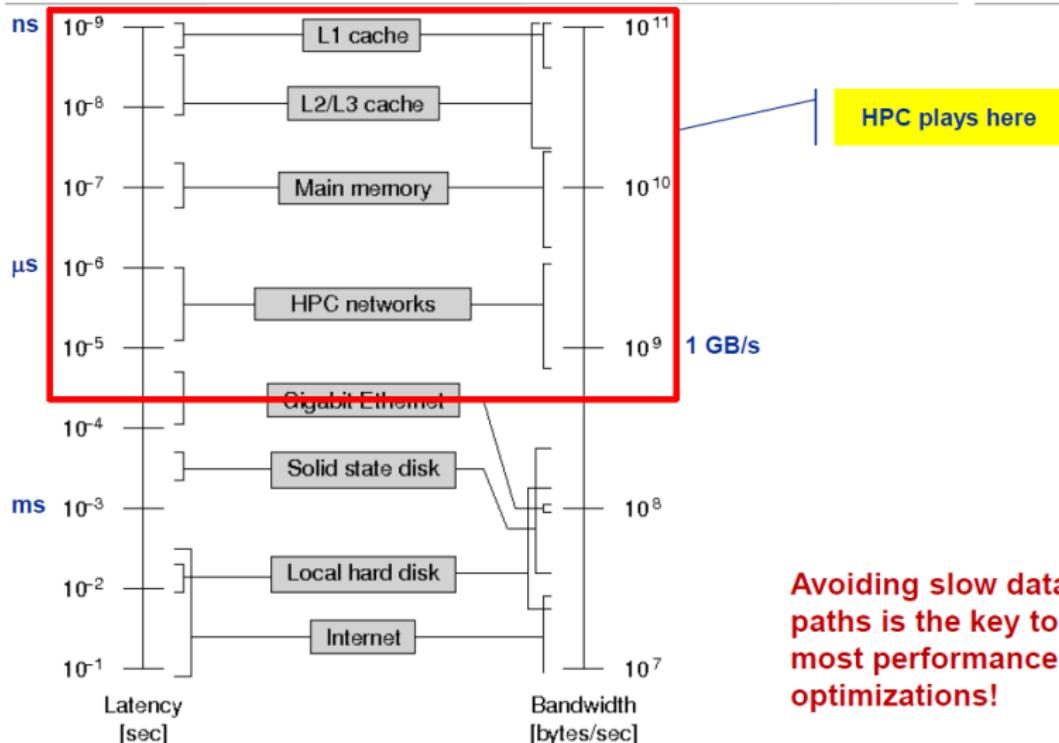


Figure: source: multicore tutorial (SC12) by Georg Hager and Gerhard Wellein



Hardware: memory hierarchy - cache

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



Memory Hierarchy: Terminology

- **Hit:** data appears in some block in the upper level (example: Block X)
 - **Hit Rate:** the fraction of memory accesses found in the upper level
 - **Hit Time:** Time to access the upper level which consists of RAM access time + Time to determine hit/miss
- **Miss:** data needs to be retrieved from a block in the lower level (Block Y)
 - **Miss Rate** = $1 - (\text{Hit Rate})$
 - **Miss Penalty:** Time to replace a block in the upper level + Time to deliver the block to the processor
- Hit Time \ll Miss Penalty (500 instructions on 21264!)

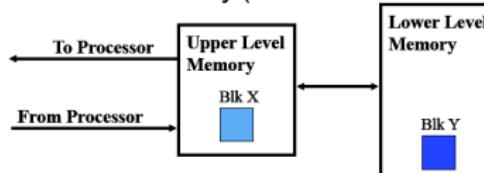
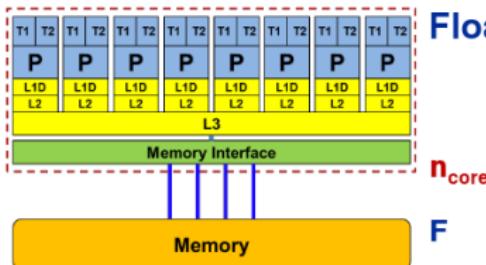


Figure: source: [T. Sterling, Louisiana State University, SC12 Tutorial](#)



Hardware: floating-point peak performance on multicore CPU



Floating Point (FP) Performance:

$$P = n_{\text{core}} * F * S * v$$

n_{core} number of cores: 8

F FP instructions per cycle: 2
(1 MULT and 1 ADD)

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

S FP ops / instruction: 4 (dp) / 8 (sp)
(256 Bit SIMD registers – “AVX”)

v Clock speed : ~2.7 GHz

TOP500 rank 1 (1995)

$$P = 173 \text{ GF/s (dp)} / 346 \text{ GF/s (sp)}$$

But: $P=5 \text{ GF/s (dp)}$ 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

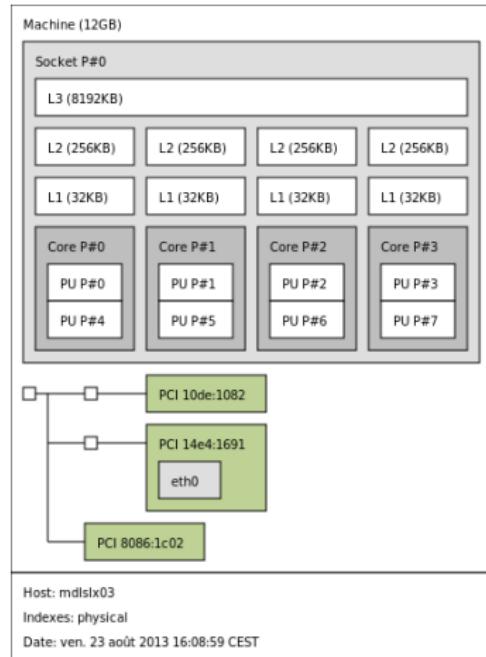
```
Fichier Édition Affichage Rechercher Terminal Aide
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 42
model name : Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
stepping : 7
microcode : 0x1a
cpu MHz : 1600.000
cache size : 8192 KB
physical id : 0
siblings : 8
core id : 0
cpu cores : 4
apicid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

- (sudo) lspci
- hwloc-ls / lstopo ([hwloc](#) - hardware locality), outil utilisé par MPI pour le placement de tâche
- lshw et lshw-gtk ([hardware LiSter](#))



Hardware: linux tools to probe hardware features

- cat /proc/cpuinfo | /bin/egrep 'processor|model name|cache size|core|siblings|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



²Un article sur [hwloc](#)

Caractéristiques d'un code dit HPC

- Code qui est exécuté efficacement sur un super-calculateur:
 - ⇒ 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 possible 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 libraries: e.g. Kokkos³, RAJA⁴, ...
- 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.pdf

³A tutorial about kokkos: [GTC2016 kokkos tutorial](#).

⁴RAJA overview [article](#).



Supercomputing trends

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

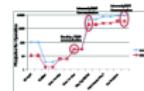
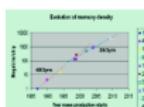
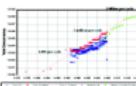
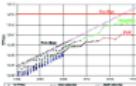
Current programming systems have WRONG optimization targets

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*



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 constraints

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](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.

