A1-1 - Aspects théoriques et algorithmiques du calcul réparti

Introduction to Parallel Programming with MPI

Master Modélisation et Simulation - ENSTA cours A1-1

Pierre Kestener - pierre.kestener@cea.fr
Edouard Audit - edouard.audit@cea.fr

CEA-Saclay, DSM, France
Maison de la Simulation

ENSTA, September, 2014

Déroulement du cours A1-1 - Introduction MPI

- cours (12/09): Introduction au HPC
- cours+TP (19/09): Initiation MPI 1
- cours+TP (26/09): Initiation MPI 2
- TP (03/10, 10/10 et 17/10): Mini-projet noté - parallélisation d'un problème de volumes finis

Les ressources de calcul utilisées pendant les TP:

- poste de travail
- cluster `gin` (ENSTA/UMA)

Recommended reading on MPI / Parallel Programming

- 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
  - An Introduction to Parallel Programming by Peter Pacheco, Morgan-Kaufmann, 2011
- on-line:
  - Cours MPI de l'IDRIS
  - Jeff Squyres's blogs on MPI and HPC
  - Parallel computing tutorial at LLNL

Introduction

- **Objectifs de ce cours:**
  - Pas juste une introduction à MPI
  - 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, mini-projet de parallélisation avec MPI
  - Outils d'analyse de performance et d'aide à la parallélisation
  - Certains sujets seront survolés, mais des pointeurs externes pour approfondir seront fournis

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)

Parallel Computing

One woman can make a baby in 9 months.

Can 9 woman make a baby in 1 month?

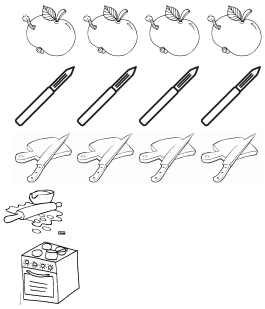
But 9 women can make 9 babies in 9 months.

Figure : source: John Urbanic, Pittsburgh Supercomputing Center

Parallel Computing: *devinette cuisine*

Devinette #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

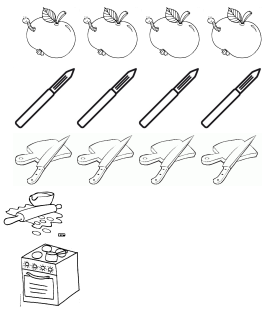
---

xx minutes à 2 ?

Parallel Computing: *devinette cuisine*

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

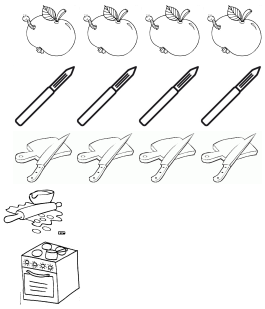
---

xx minutes à 4 ?

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

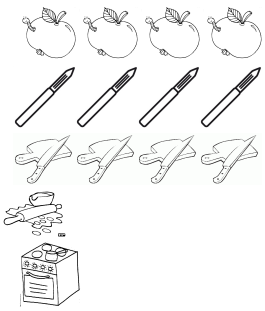
---

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

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. . .

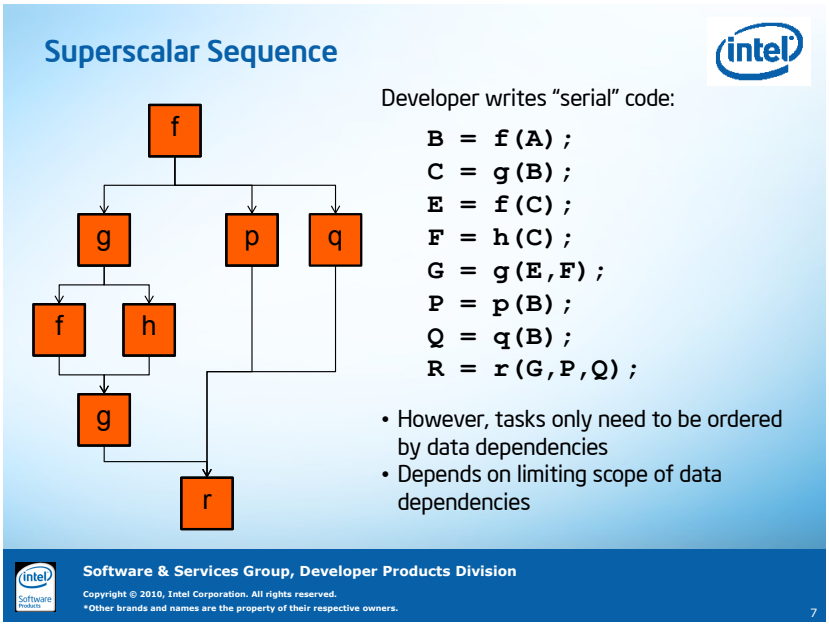
Parallel Computing: definition(s)

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

Parallel Computing: definition(s)

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).

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 within bounded time
  - Save on Wall clock time
  - Overcome memory constraints
  - Utilize non-local resources

source: M. Zahran, NYU

Pourquoi paralléliser ?

- **When to Parallelize:**
  - 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, ...

Code / Logiciel

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

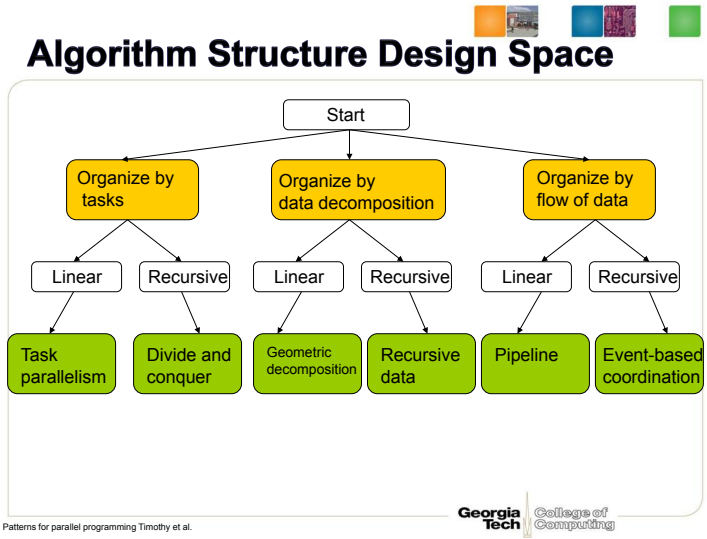
- Conception d'un code parallèle
- Paralléliser les données ?
  - Modèle à mémoire distribuée, ex: MPI
  - Modèle à mémoire partagée (SMP), ex: OpenMP, pthread, ..
  - nouvelles problématiques, ex: cohérence de cache
- Paralléliser l'algorithme
  - Transformée de Fourier
  - Problème à N-corps
- Hardware
  - 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
- Paralléliser les données ?
  - Modèle à mémoire distribuée, ex: MPI
  - Modèle à mémoire partagée (SMP), ex: OpenMP, pthread, ..
  - nouvelles problématiques, ex: cohérence de cache
- Paralléliser l'algorithme
  - Transformée de Fourier
  - Problème à N-corps
- 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

(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)

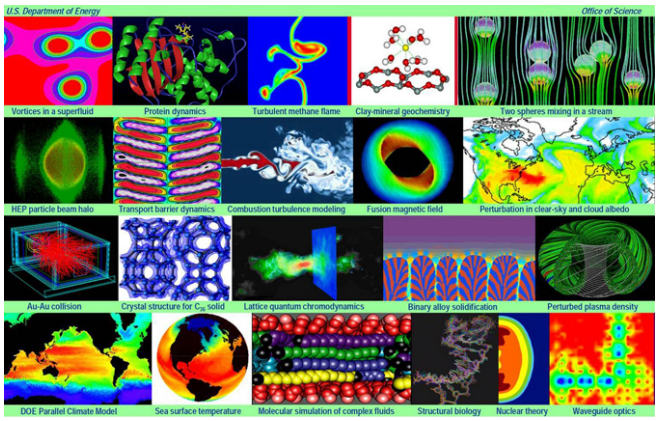


Figure : source: http://www.scidac.gov/

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

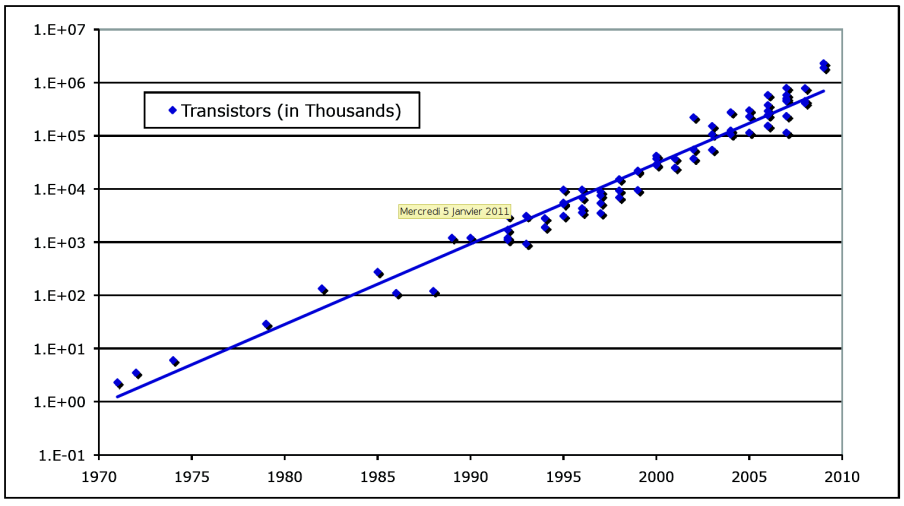
Strategic importance of supercomputing:

- 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...*

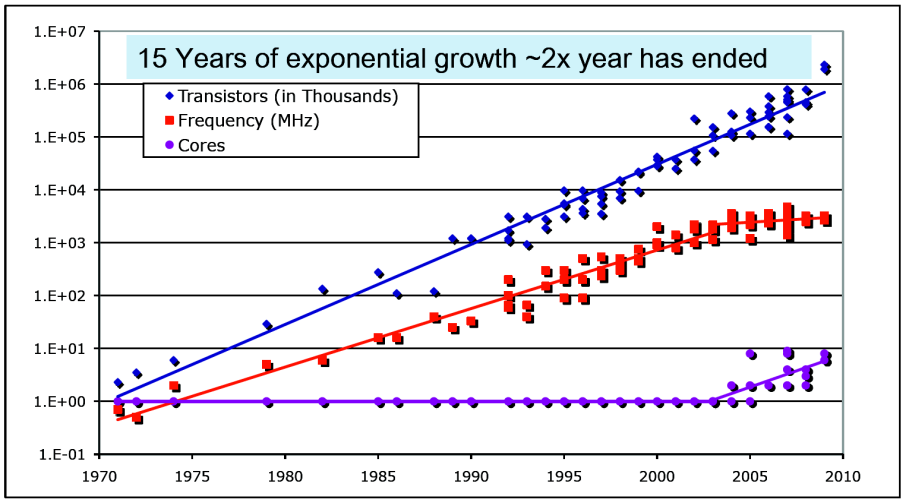
The number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years



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.

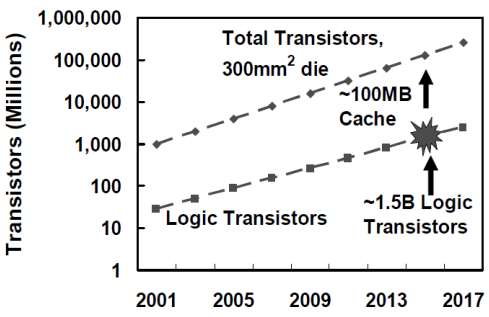


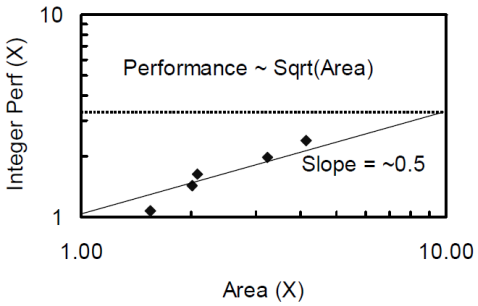
Figure 1: Transistor integration capacity

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

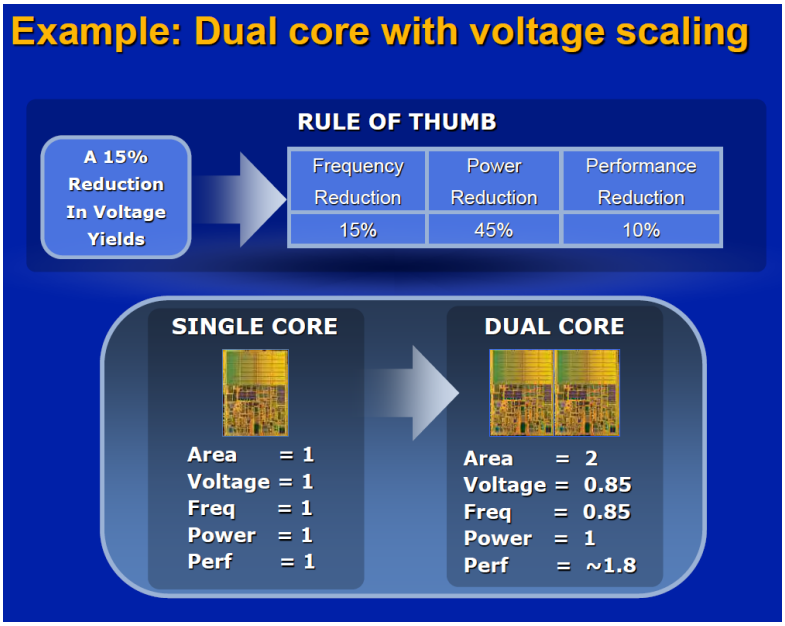
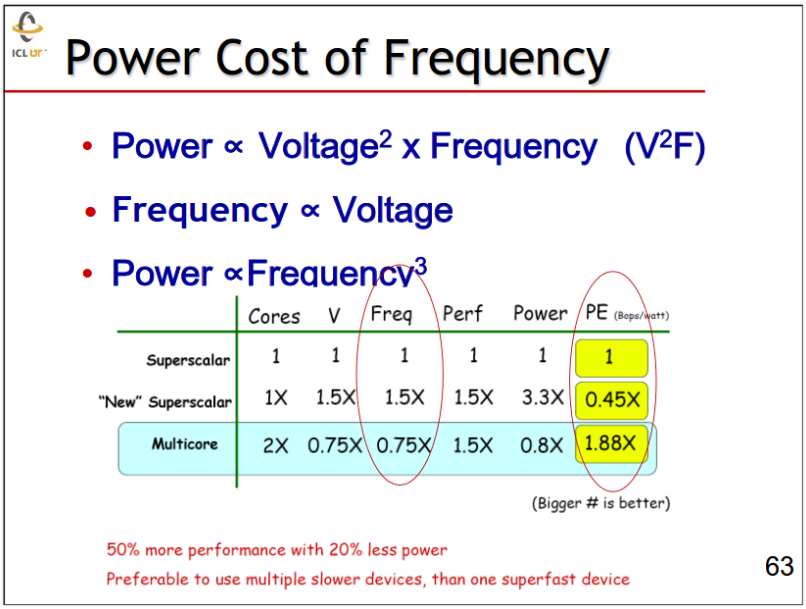


Figure : source: John Urbanic, Pittsburgh Supercomputing Center

Moore's law - Towards multi-core architectures



Power Cost of Frequency

- Power ∝ Voltage^2 x Frequency (V^{2}F)
- Frequency ∝ Voltage
- Power ∝ Frequency^3

	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)

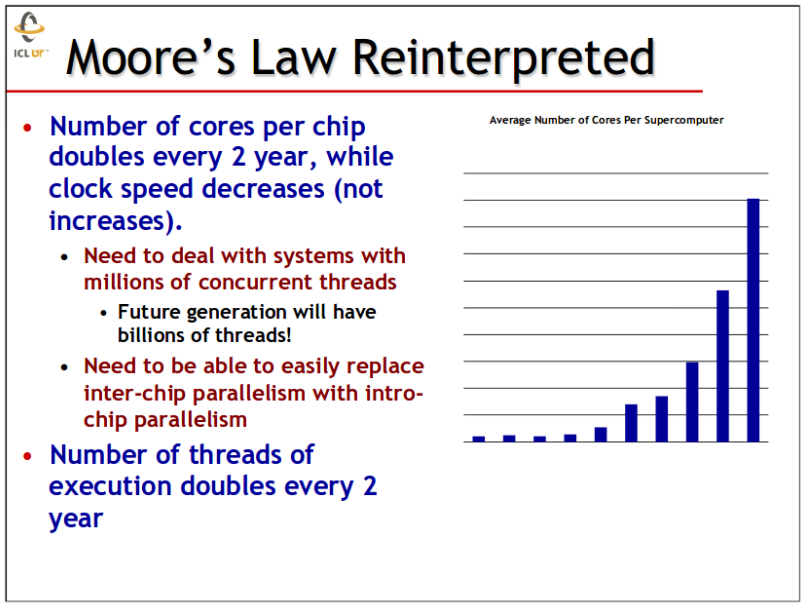
50% more performance with 20% less power

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 intro-chip parallelism
- Number of threads of execution doubles every 2 year

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

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

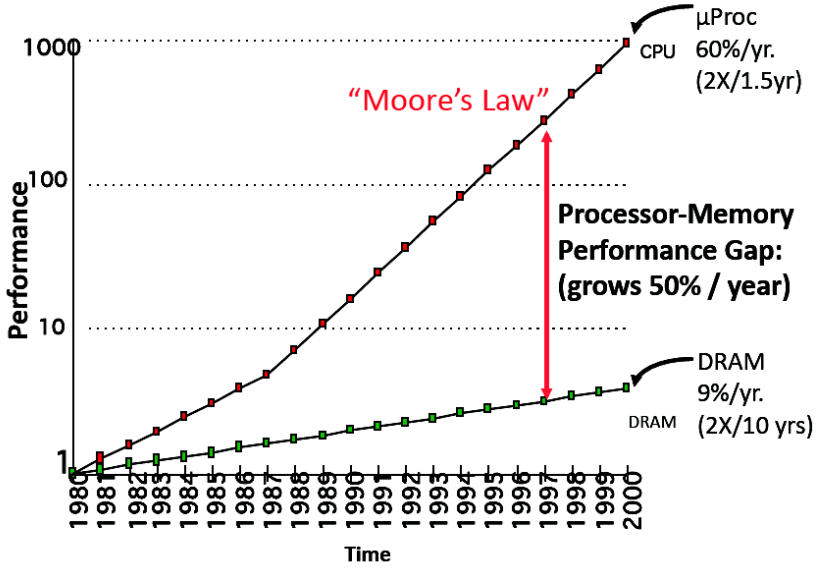


Figure : source: T. Sterling, Louisiana State University

Progrès algorithmiques

Whence new algorithms?

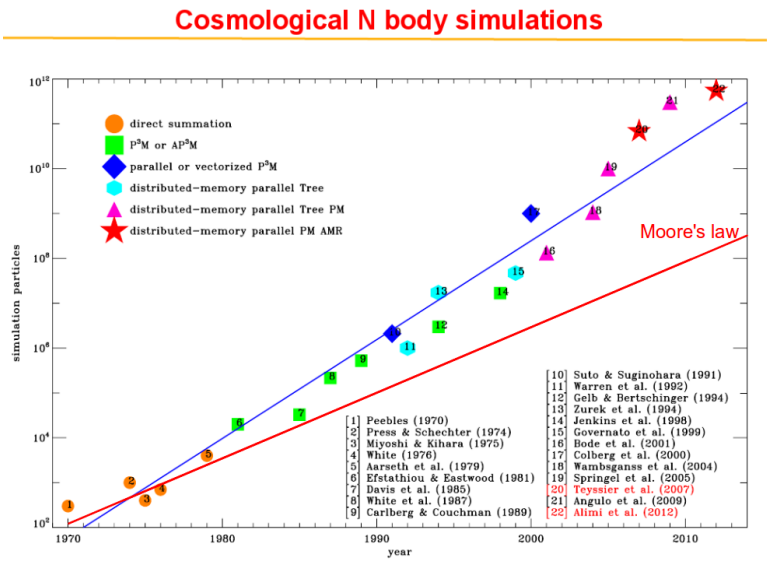
- Algorithms arise to fill the gap between *architectures that are available* and *applications that must be executed*
- Many algorithmic advances are oriented towards particular physical problems that defy the assumptions of today's optimal methods – e.g., *anisotropy, inhomogeneity, geometrical irregularity, 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



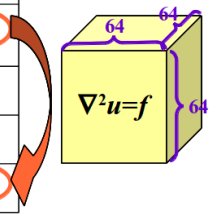
source: R. Teyssier

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



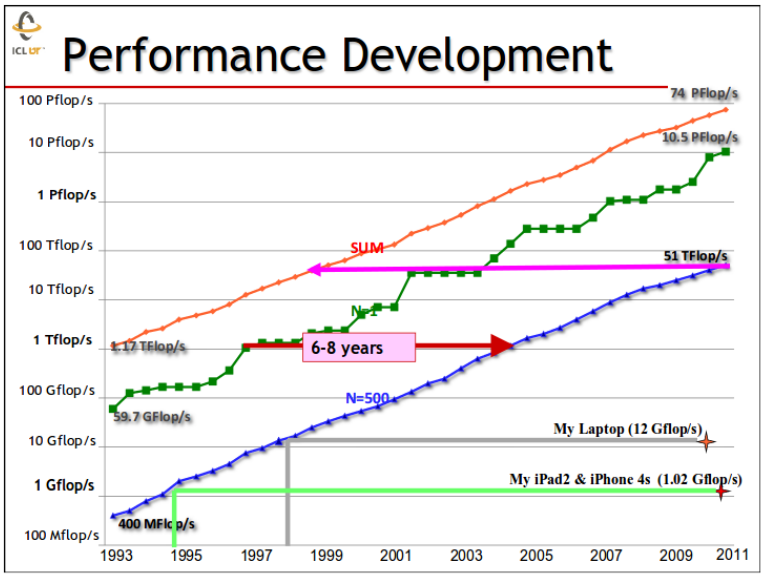
- If $n=64$, this implies an overall reduction in flops of ~16 million*

*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*

"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

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

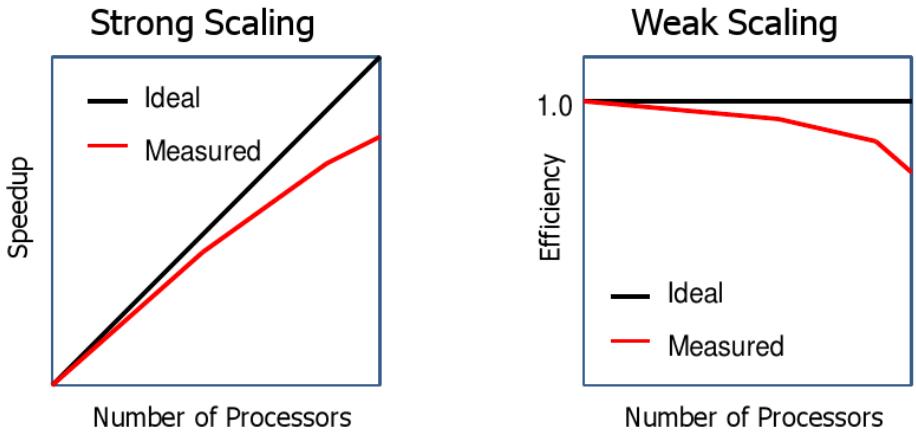
  ideallement: N

- **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.

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 calculateurs dont la puissance de calcul est proche des limites de la techonologie 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 (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 ?

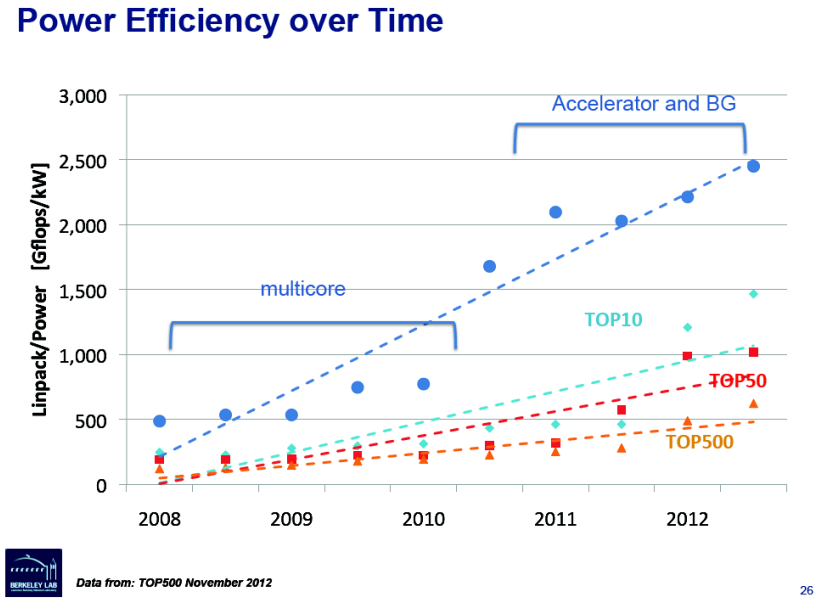


Figure : Horst Simon, LBNL

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
- 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

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, ...

- 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'éxecution*: `time`, *gettimeofday*
- *Profiling*: `gprof`, perf, PAPI
- *Tracing*: TAU, scalasca, scoreP

Supercomputers / Mesure de performance MPI

- MPIBlib: P2P, collective MPI communication benchmark

Von Neumann architecture

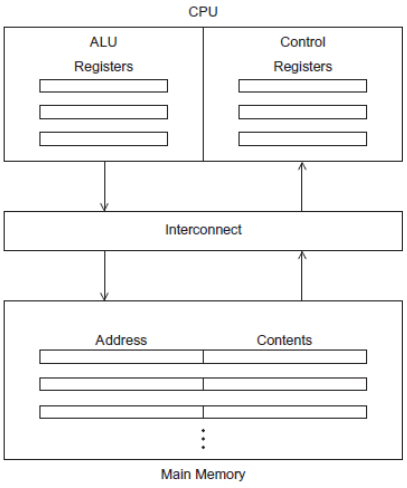


Figure : Peter Pacheco, SanFrancisco U.

Multi-core CPU

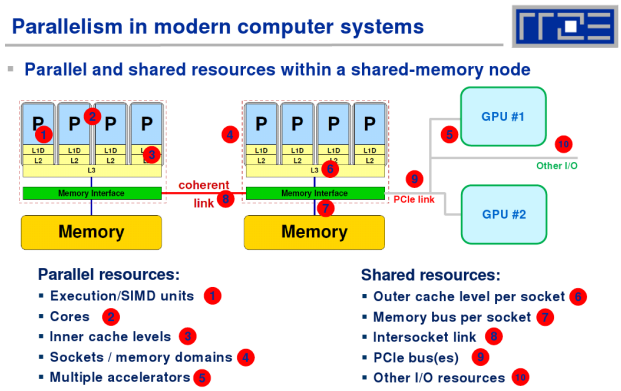


Figure : source: multicore tutorial (SC12) by Georg Hager and Gerhard Wellein

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

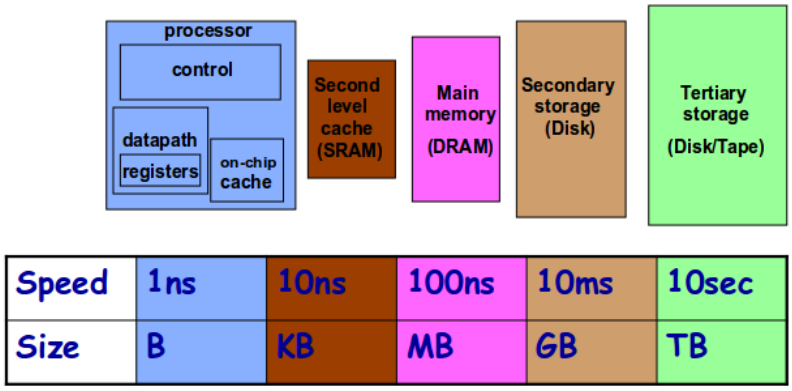


Figure : source: Scientific Computing for engineers, CS594, J. Dongarra

Hardware: memory hierarchy - low / high latency

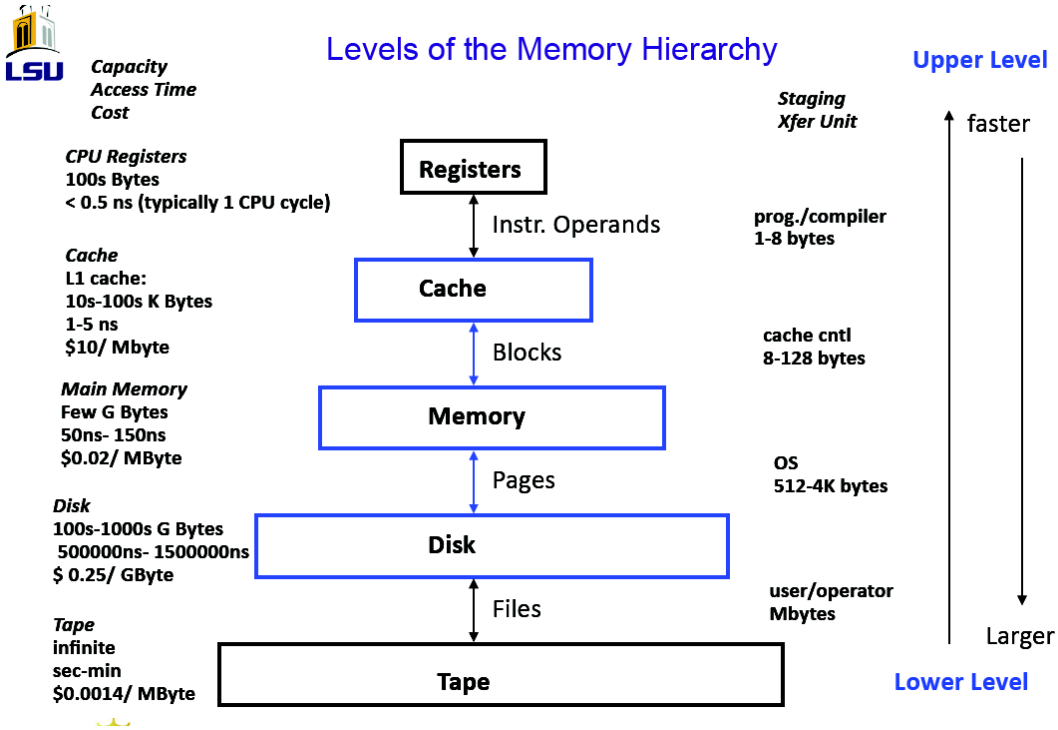


Figure : source: T. Sterling, Louisiana State University, SC12 Tutorial

Hardware: memory hierarchy - low / high latency

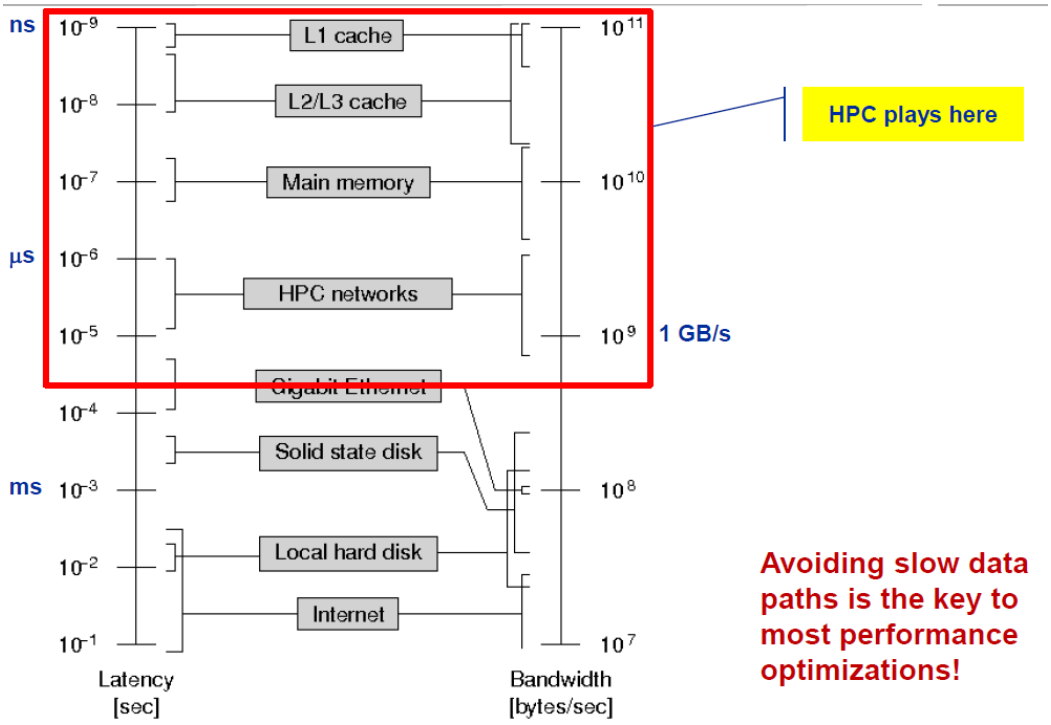


Figure : source: multicore tutorial (SC12) by Georg Hager and Gerhard Wellein

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

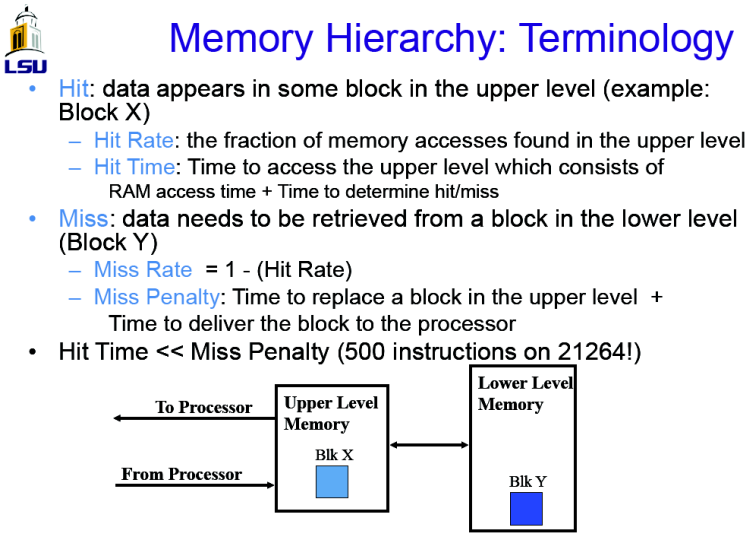


Figure : source: T. Sterling, Louisiana State University, SC12 Tutorial

Hardware: floating-point peak performance on multicore CPU

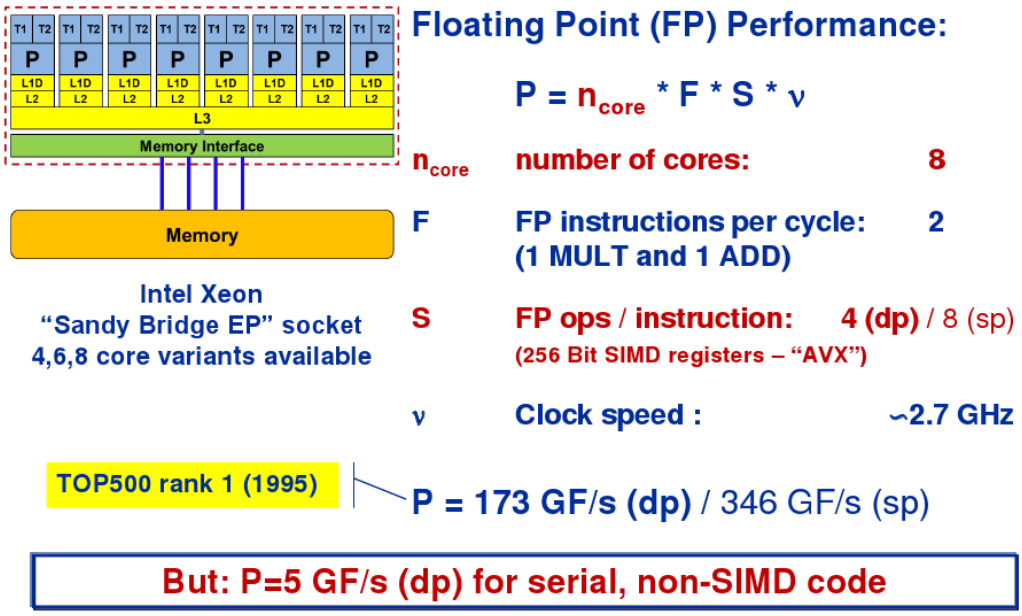


Figure : source: multicore tutorial (SC12) by Georg Hager and Gerhard Wellein

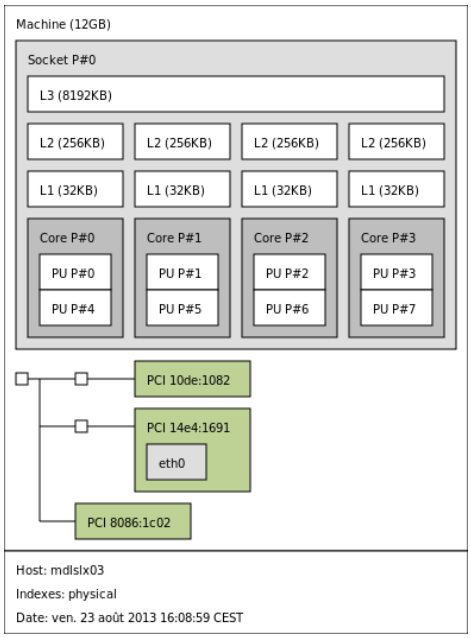
Hardware: linux tools to probe hardware features

- `cat /proc/cpuinfo | /bin/egrep 'processor|model name|cache size|core|sibling|physical'`
  - numérotation attribuée par le noyau linux
  - `core id`: numero d'un cœur de CPU
  - `physical id`: numero 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`: numero d'un cœur de CPU
  - `physical id`: numero 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 `gin` (environnement: `module load hwloc`) et retrouver les caractéristiques des nœuds de calcul

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

Le cluster de l'UMA: machine `gin`

Les nœuds de calcul

- **1 frontale interactive** `gin.ensta.fr` équipées de :
  - 2 processeurs - quadcore - AMD Opteron 2378
  - 8 Go de mémoire
  - utilisation: compilation des applications; lancer les jobs
- **31 nœuds de calculs** `gin[1-31]` hétérogènes:

Le cluster de l'UMA: machine `gin`

Autres caractéristiques:

- pas de système de fichiers parallèle
- Réseau: infiniband QDR
- Système d'exploitation: Rocks 5.5 (dérivé CentOS)
- Gestionnaire des ressources / tâches (*jobs*): GridEngine (SGE)

connection depuis la salle de cours:

- tappez simplement `ssh gin`. Vous êtes alors connecté sur la *frontale* de `gin`

Cluster GIN (ENSTA / UMA) - environnement logiciel

Environnement module (http://modules.sourceforge.net/):

La gestion de l'environnement est faite via l'utilisation de `module` :

- Liste les softs disponibles :
  `module avail`
- Liste les softs intégrés /chargés dans l'environnement courant :
  `module list`
- Intègre/charge le logiciel `tool` dans l'environnement courant :
  `module load tool/version`
- Supprime le soft tool de l'environnement courant :
  `module unload tool/version`
- Remplace la version *old-version* du soft `tool` par la version *new-version* dans l'environnement courant :
  `module switch tool/old-version tool/new-version`
- Affiche les variables d'environnement modifiées/restaurées quand on charge/décharge un module :
  `module show tool/version`

Cluster GIN (ENSTA / UMA) - gestionnaire de travaux

- **GridEngine (ex. SGE)** de la distrib Rocks.
- Définition d'un ***job***: un script shell, qui contient les commandes à exécuter (e.g. `mpirun`) et des commentaires commençant par `#$` permettant de spéficier les ressources nécessaires à l'exécution du travail (e.g. nombre de tâches MPI, nombre de nœuds de calcul, mémoire, temps maximal d'exécution, etc...)
- les commandes utiles de base:
  - `qsub`: soumettre un travail au calculateur. Une fois dans la queue des travaux, le travail reçoit un numéro (*jobId*)
    `qsub job.qsub`
  - `qstat`: liste des travaux en cours, et leur état (en exécution, en attente des ressources disponibles, etc...)
    `qstat`; `qstat -j` *jobId*
  - `qdel`: enlever/tuer un job de la queue
    `qdel` *jobId*
  - `man sge_intro` pour plus d'information.
  - Voir aussi http://en.wikipedia.org/wiki/Oracle_Grid_Engine

Cluster GIN (ENSTA / UMA) - soumission de travaux

- script de soumission

```
#!/bin/bash
#
#  Name of the job (used to build the name of the standard output
     stream)
#$ -N test_job
#
#  Number of MPI task requested
#$ -pe orte 4
#
#  The job is located in the current working directory
#$ -cwd
#
#  Merge standard error and standard output streams
#$ -j y
#

mpirun -bycore -bind-to-core -report-bindings ./helloworld
```

../code/submit_job_gin.qsub

Cluster GIN (ENSTA / UMA) - Grid Engine -qstat

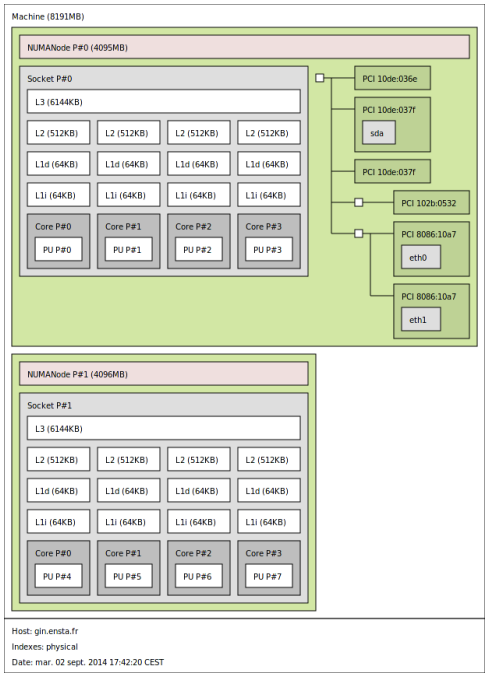
Exemple de sortie de la commande qstat:

```
kestener@gin:~/coursMPI/cours_MPI_code/c$ qstat

job-ID  prior   name       user         state submit/start at     queue      slots
------------------------------------------------------------------------------------
     11 0.00000 test_job   kestener     qw    09/02/2014 15:53:17                 4
```

qstat_output.txt

Topologie de la frontale de GIN



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!

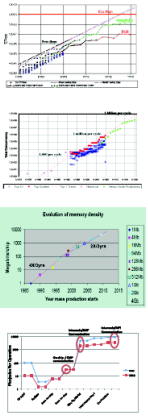
Figure : source: multicore tutorial (SC12) by Georg Hager and Gerhard Wellein

Supercomputing trends

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

Current programming systems have WRONG optimization targets

Old Constraints	New Constraints
• **Peak clock frequency** *as primary limiter for performance improvement*	• **Power** *is primary design constraint for future HPC system design*
• **Cost:** *FLOPs* are biggest cost for system: *optimize for compute*	• **Cost:** *Data movement dominates: optimize to minimize data movement*
• **Concurrency**: Modest growth of parallelism by adding nodes	• **Concurrency:** *Exponential growth of parallelism within chips*
• **Memory scaling***: maintain byte per flop capacity and bandwidth*	• **Memory Scaling:** *Compute growing 2x faster than capacity or bandwidth*
• **Locality***: MPI+X model (uniform costs within node & between nodes)*	• **Locality***: must reason about data locality and possibly topology*
• **Uniformity**: *Assume uniform system performance*	• **Heterogeneity**: *Architectural and performance non-uniformity increase*
• **Reliability***: It's the hardware's problem*	• **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 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.