Éléments de parallélisme et parallélisation automatique pour GPU et moulticœurs

M2 MIHP / ECP + UVSQ

Ronan KERYELL[3] (rk@hpc-project.com)

HPC Project

—

9 Route du Colonel Marcel Moraine[1]
92360 Meudon La Forêt, France

—

Rond Point Benjamin Franklin[2]
34000 Montpellier, France

—

5201 Great America Parkway #3241[3]
Santa Clara, CA 95054, USA

12/01/2012

Outline

Outline

Modéliser le monde

- Vieux rêve de l'humanité : mieux comprendre le monde
- Développement des mathématiques pour comprendre
- Modélisation du monde dans un formalisme mathématique pour prévoir
- Systèmes complexes $\rightsquigarrow$ rarement des solutions analytiques
- Automatisation des calculs lents & immenses, sujets aux erreurs

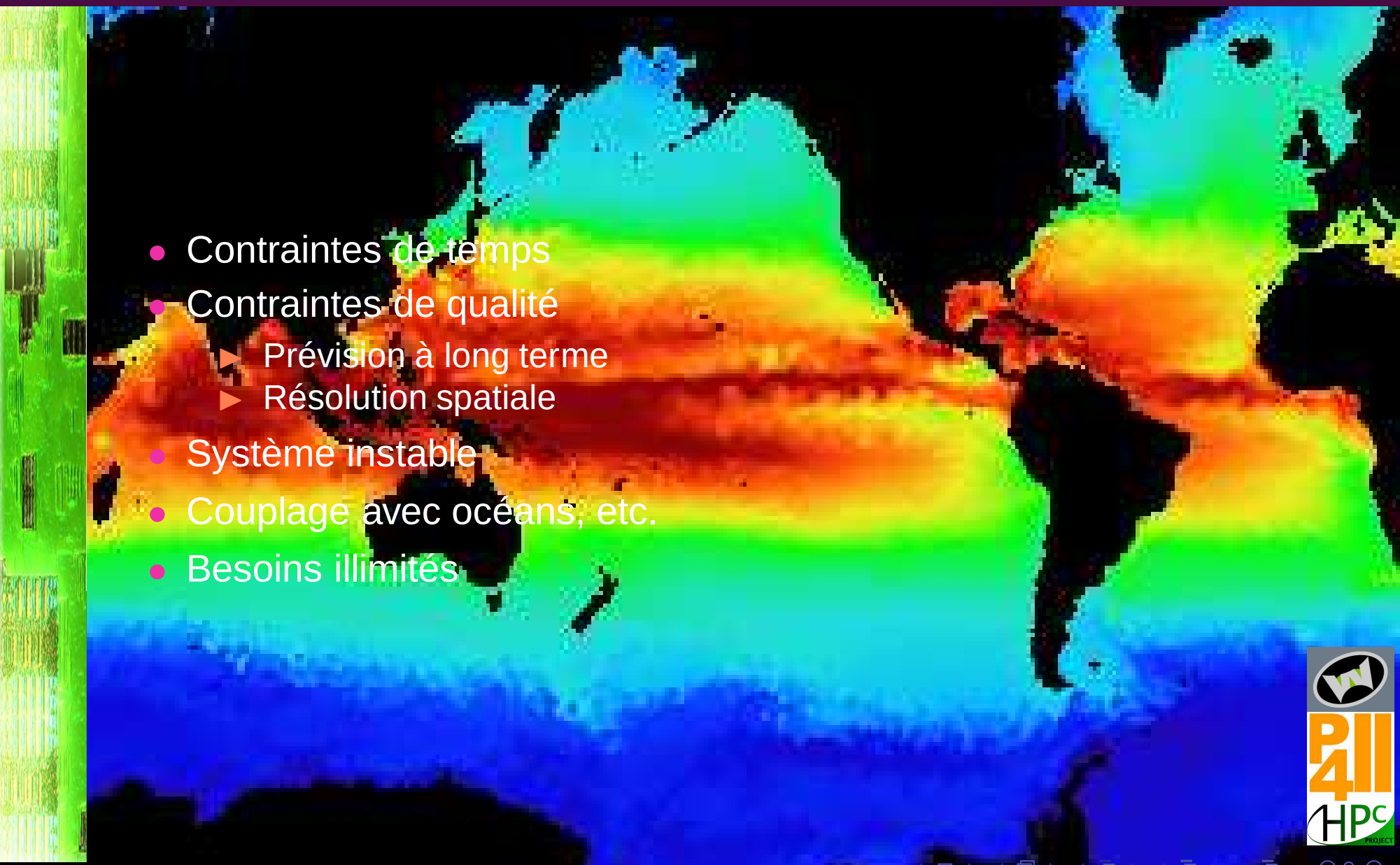
Développement d'ordinateurs pour :

- Gros modèles : prévisions plus précises
- Automatisation de tâches (gestion)
- Nécessité de résultats rapidement

Quelques applications

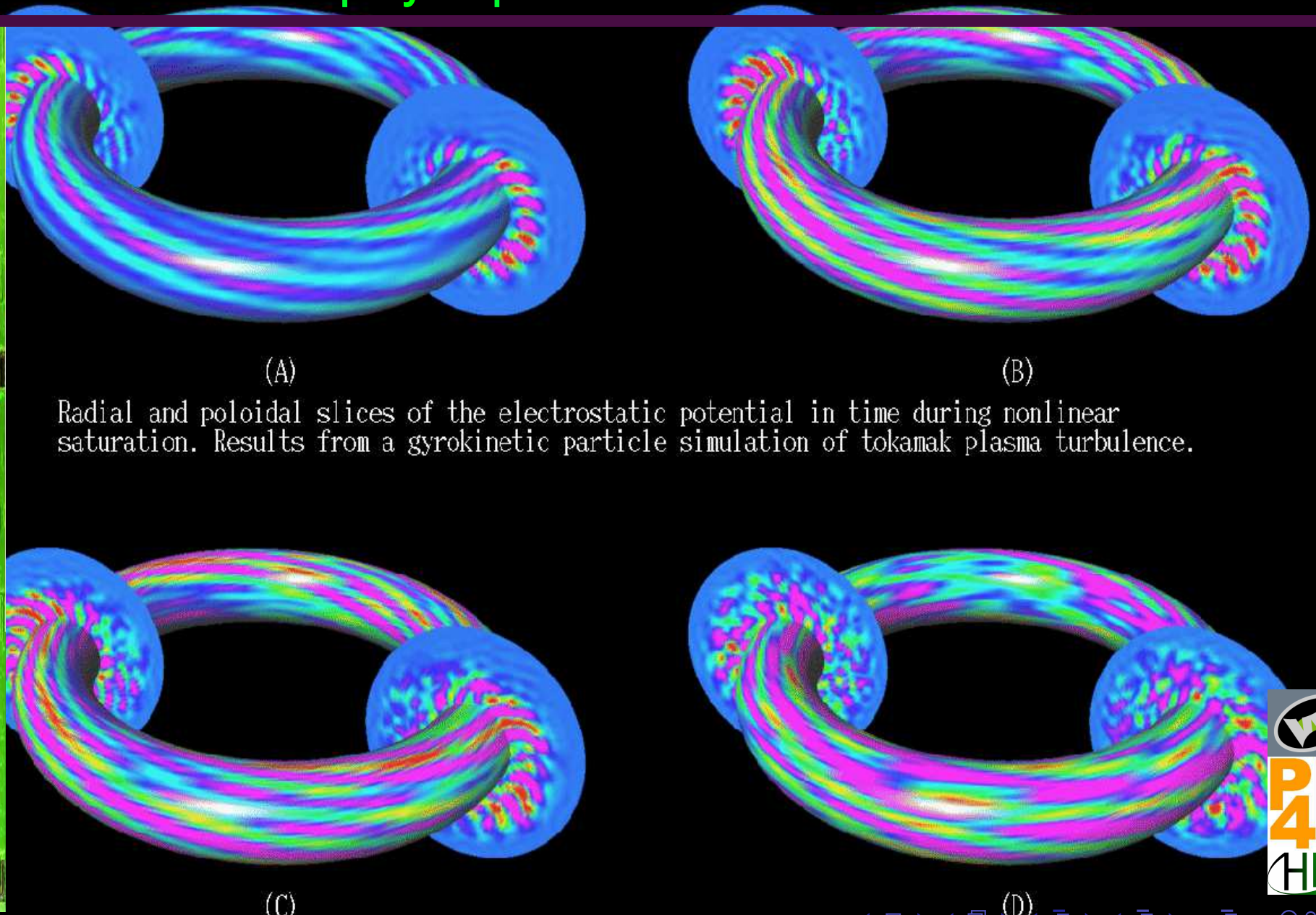
- Simulation et calcul numérique, expérimentation numérique
  - Environnement
  - Biologie moléculaire
  - Conception industrielle (voitures, clubs de golf...)
  - Chimie *ab initio*
  - Simulations militaires (moratoire nucléaire, ASCI Accelerated Strategic Computing Initiative aux US...)
- Grosses bases de données & *data-mining*, serveur WWW, réseaux sociaux...
- Simulations financières

Prévision météorologique



- Contraintes de temps
- Contraintes de qualité
  - Prévision à long terme
  - Résolution spatiale
- Système instable
- Couplage avec océans, etc.
- Besoins illimités

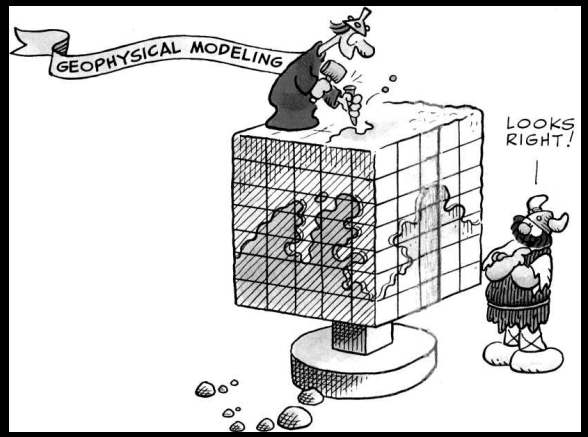
Tokamak — physique nucléaire



Radial and poloidal slices of the electrostatic potential in time during nonlinear saturation. Results from a gyrokinetic particle simulation of tokamak plasma turbulence.

Contraintes sur les ordinateurs

- Plus d'informations à stocker (maillages + fins) ↗ mémoire



- Plus de calculs à faire, ↗ vitesse de calcul
  - Débit : beaucoup de calculs aboutissent/unité de temps
  - Latence : temps d'exécution d'une tâche
- Applications limitées par les performances
- Désirs : toujours plus !

  μηδὲν ἄγαν
- Mesures par programmes étalons (*Benchmark*)

Ordinateurs les plus rapides d'une époque (1–4 ordres de grandeur) :

supercalculateurs

Gros gains aussi sur les algorithmes...

Outline

Top 500

`http://top500.org`

- Liste 500 plus gros ordinateurs déclarés dans le monde depuis 1993
- Top 10 : crème de la crème
- Étalon : factorisation de matrice *LU* LINPACK
  - Plus de calculs que de communications
  - Cas d'école hyper régulier rarement rencontré dans la vraie vie
  -  À considérer comme une puissance crête (efficace)

Permet d'estimer les directions futures technologiques de l'informatique « standard »

Top 10 — November 2011

TFLOPS performance

Rank	*Site*	*Computer/Year Vendor*	*Cores*	*Rmax*	*Rpeak*	Power (kW)
1	RIKEN Advanced Institute for Computational Science (AICS), Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu	705 024	10510	11280	12660
2	National Supercomputing Center in Tianjin, China	Tianhe-1A - NUDT YH MPP, X5670 2.93 Ghz 6C, NVIDIA C2050 / 2010 (NUDT)	186 368	2566	4701	4040
3	DOE/SC/Oak Ridge National Laboratory, USA	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 (Cray Inc.)	224 162	1759	2331	6950
4	National Supercomputing Centre in Shenzhen (NSCS), China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVIDIA Tesla C2050 GPU, Infiniband QDR / 2010 (Dawning)	120 640	1271	2984	2580
5	GSIC Center, Tokyo Institute of Technology, Japan	TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, NVIDIA GPU, Linux/Windows / 2010 (NEC/HP)	73 278	1192	2287	1398
6	DOE/NNSA/LANL/SNL, USA	Cray XE6 Opteron 6136 8C 2.4 GHz, custom network / 2010 (Cray Inc.)	142 272	1110	1366	3980
7	NASA/Ames Research Center/NAS, USA	SGI Altix ICE 8200EX/8400EX, Xeon HT QC 3.0/Xeon 5570/5670 2.93 Ghz, Infiniband / 2011 SGI	111104	1088	1315	4102
8	DOE/SC/LBNL/NERSC, USA	Hopper - Cray XE6 12-core 2.1 GHz custom network / 2010 (Cray Inc.)	153 408	1054	1288	2910
9	Commissariat a l'Energie Atomique (CEA), France	Tera-100 - Bull bullx super-node S6010/S6030 / 2010 (Bull SA)	138 368	1050	1254	4590
10	DOE/NNSA/LANL, USA	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband / 2009 (IBM)	122 400	1042	1375	2345

http://www.top500.org/list/2010/11/100

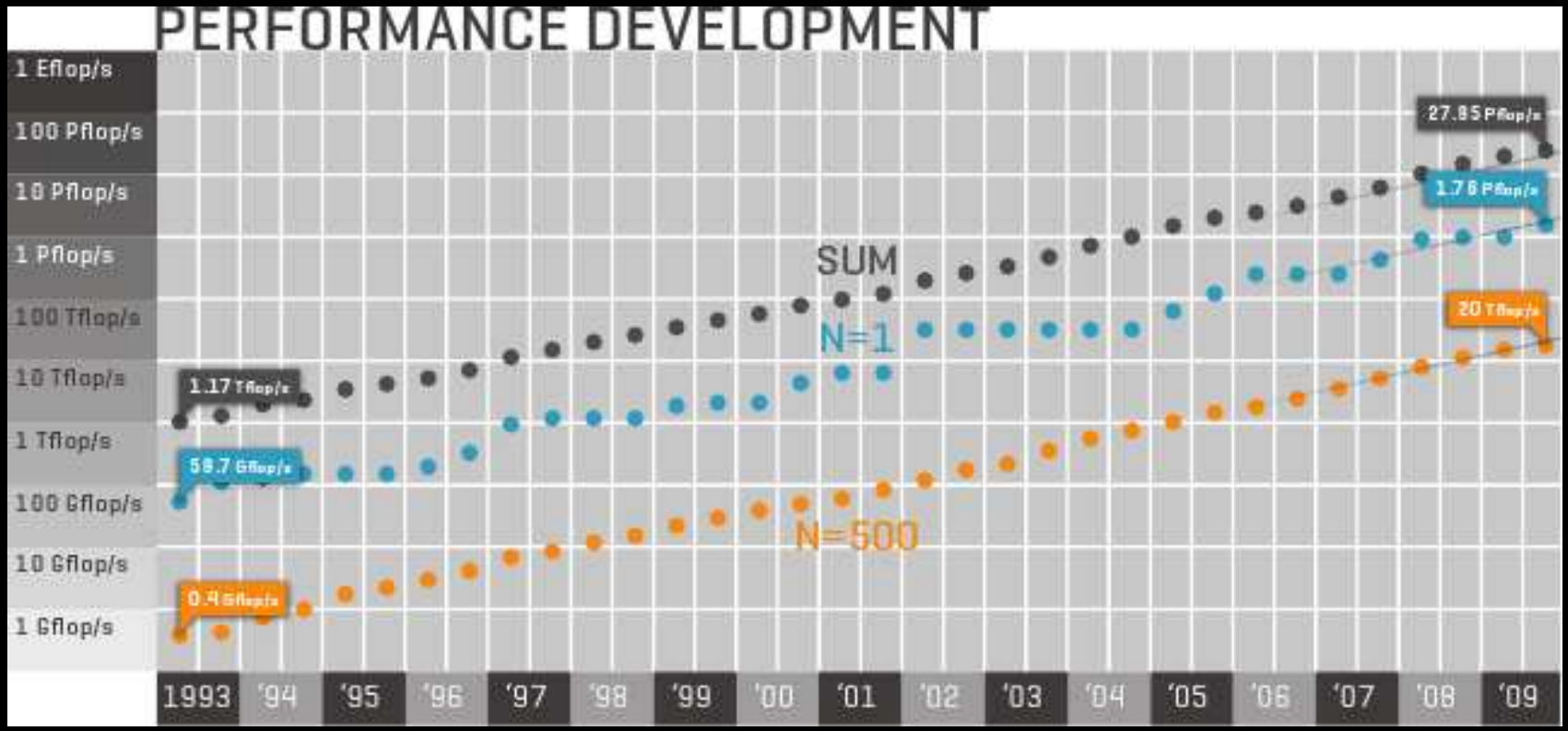
Le Top 10 — novembre 2009

Performances en TFLOPS

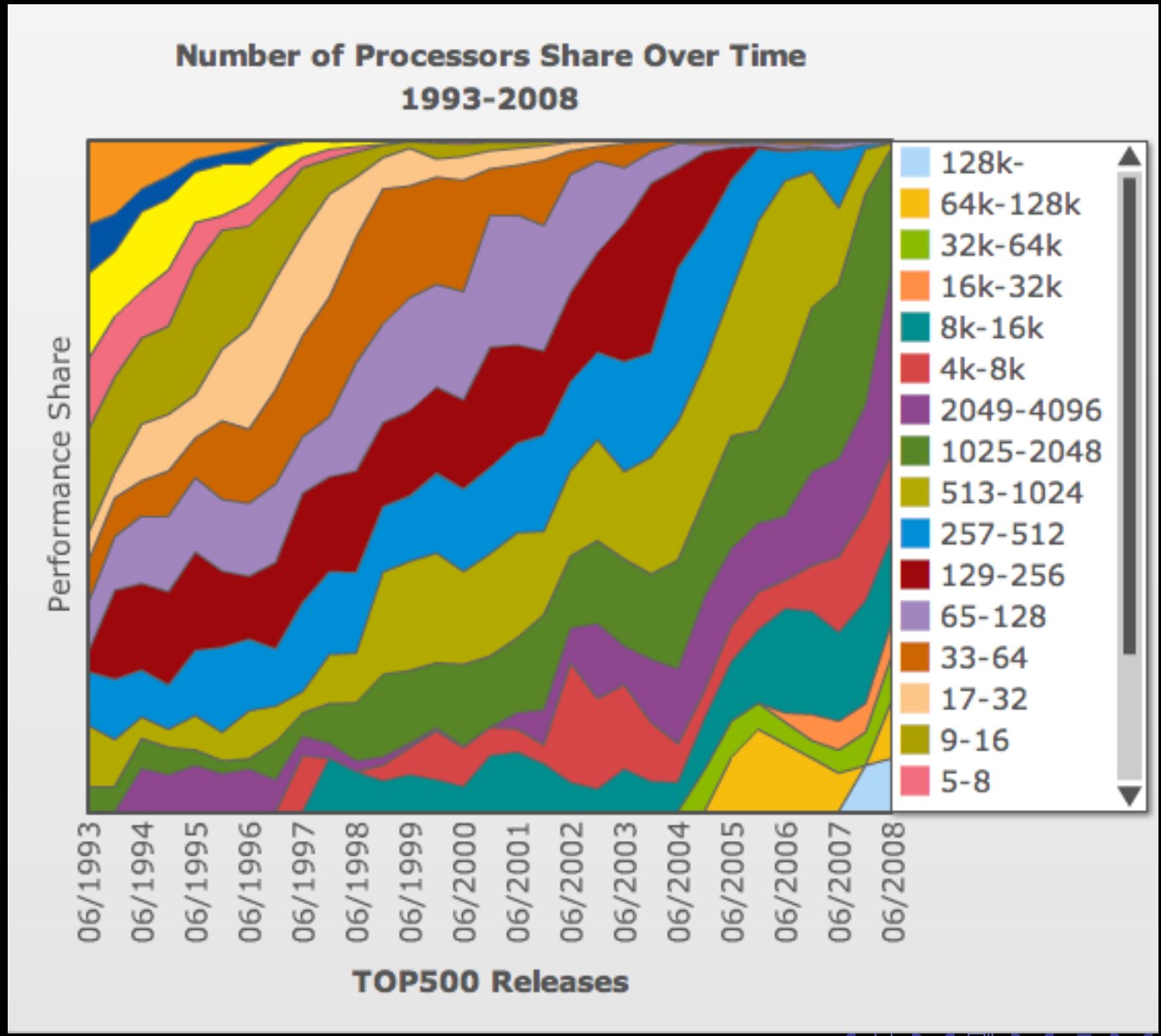
Rank	*Site*	*Computer/Year Vendor*	*Cores*	*Rmax*	*Rpeak*	Power (kW)
1	Oak Ridge National Laboratory	Cray XT5-HE Opteron Six Core 2.6 GHz	224162	1759.00	2331.00	6950.60
2	DOE/NNSA/LANL	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband	122400	1042.00	1375.78	2345.50
3	National Institute for Computational Sciences/University of Tennessee	Cray XT5-HE Opteron Six Core 2.6 GHz	98928	831.70	1028.85	
4	Forschungszentrum Juelich (FZJ)	Blue Gene/P Solution	294912	825.50	1002.70	2268.00
5	National SuperComputer Center in Tianjin/NUDT	NUDT TH-1 Cluster, Xeon E5540/E5450, ATI Radeon HD 4870 2, Infiniband	71680	563.10	1206.19	
6	NASA/Ames Research Center/NAS	SGI Altix ICE 8200EX, Xeon QC 3.0 GHz/Nehalem EP 2.93 Ghz	56320	544.30	673.26	2348.00
7	DOE/NNSA/LLNL	eServer Blue Gene Solution	212992	478.20	596.38	2329.60
8	Argonne National Laboratory	Blue Gene/P Solution	163840	458.61	557.06	1260.00
9	Texas Advanced Computing Center/Univ. of Texas	SunBlade x6420, Opteron QC 2.3 Ghz, Infiniband	62976	433.20	579.38	2000.00
10	Sandia National Laboratories / National Renewable Energy Laboratory	Sun Blade x6275, Xeon X55xx 2.93 Ghz, Infiniband	41616	423.90	487.74	
470	IFREMER	SGI Altix ICE 8200EX, Xeon X5560 quad core 2.8 GHz	2048	21345	22937.6	

http://top500.org/files/newsletter112009_tabloid_v3.pdf

Performance totale — novembre 2009



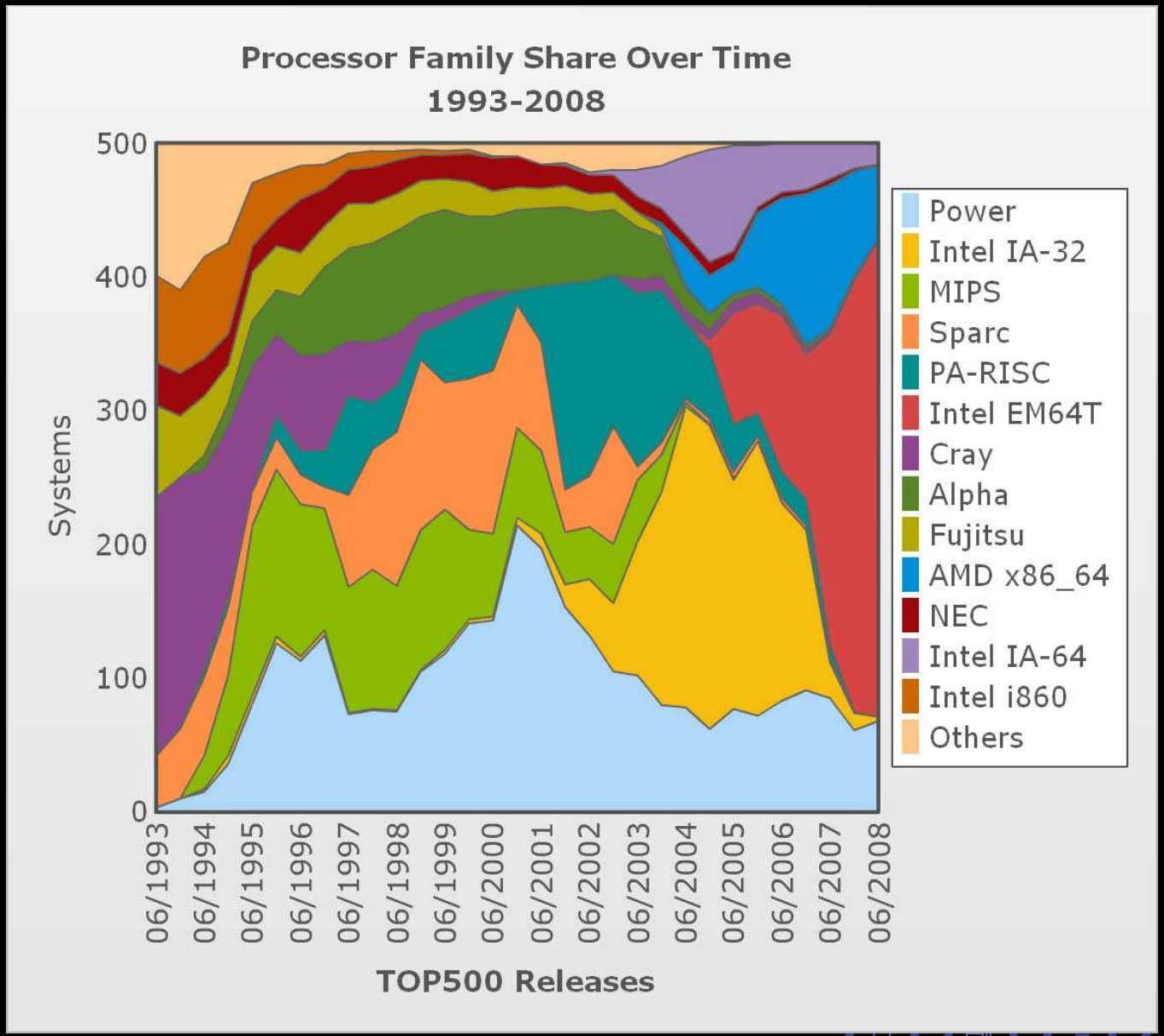
Parallélisme massif — 06/2008



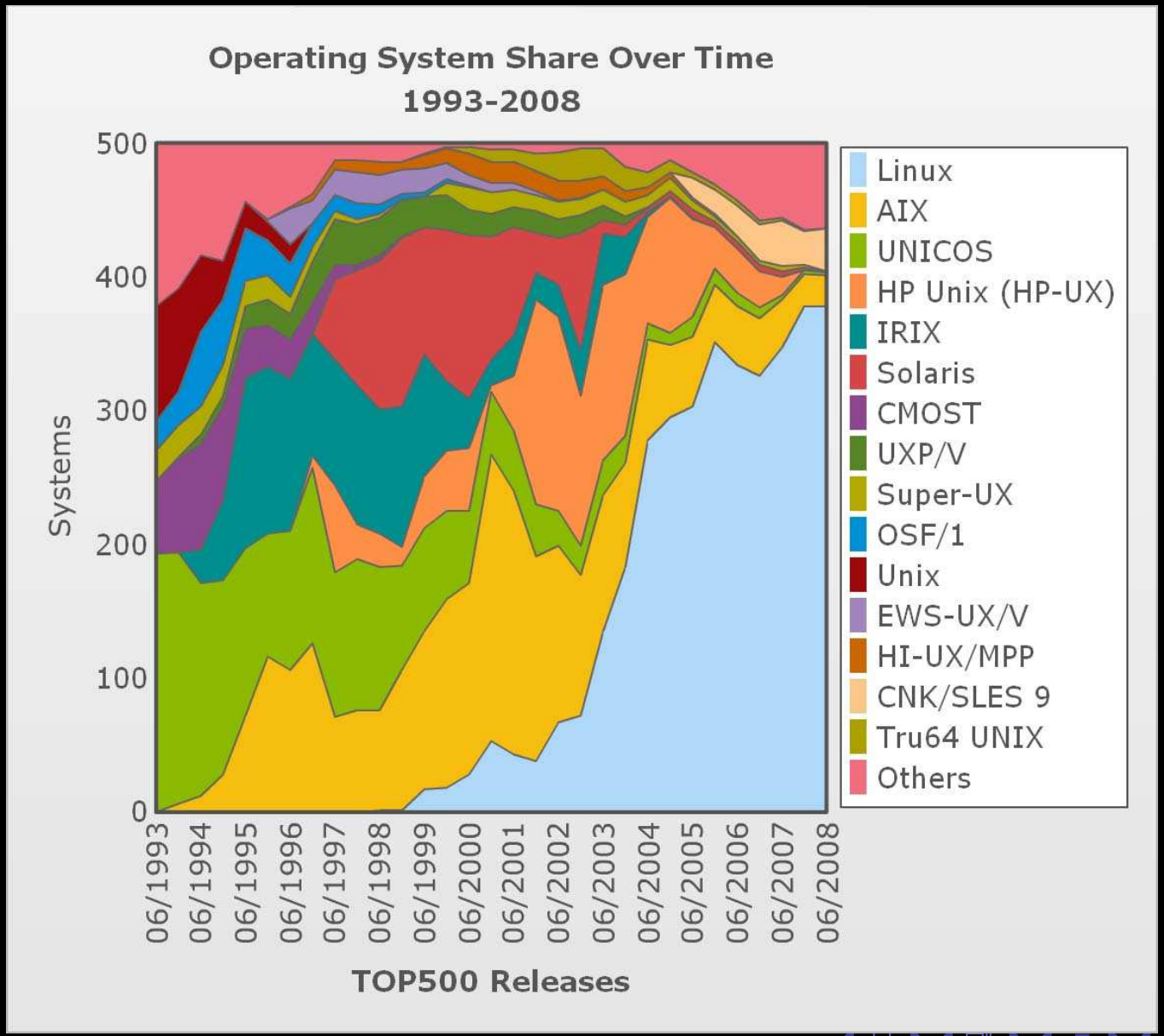
Types de machines parallèles

- Vectoriels : processeurs optimisés pour traiter des vecteurs
- Grappes (*clusters*) : ordinateurs standards reliés par réseau
- MPP (*Massively Parallel Processors*) : ordinateurs plus spécialisés intégrés dans réseau spécifique
- SMP (*Shared Memory Processor*) : processeurs qui partagent une mémoire commune (cas des ordinateurs standard multicœurs d'aujourd'hui)
- Constellations : clusters de gros SMP

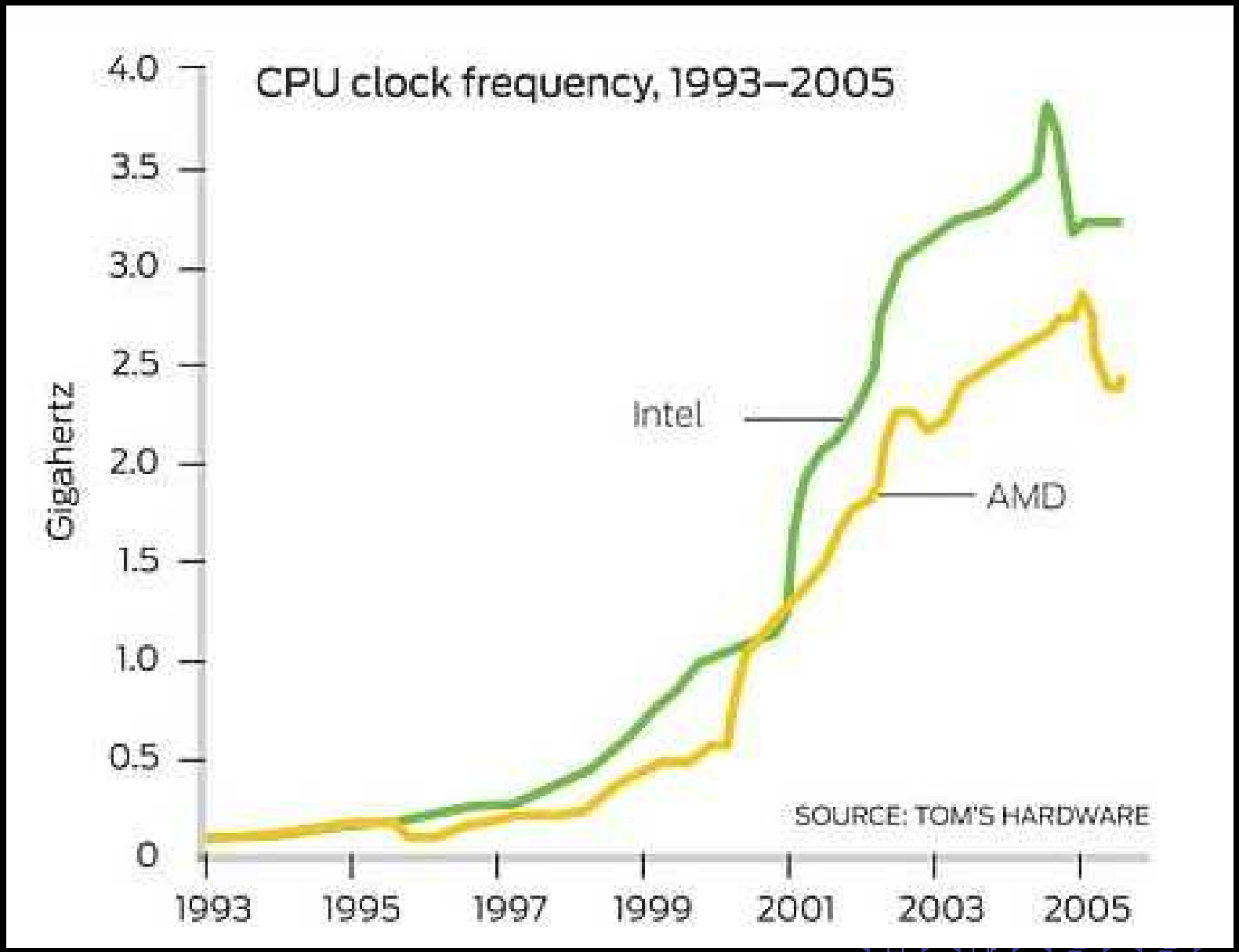
Types de processeurs — juin 2008



Types de systèmes d'exploitation — juin 2008



Évolution vitesse des processeurs



Green 500

- Problème de puissance dissipée dans les centres de calcul... ☹
- 6 MW ≈ 600 €/h, ≈ 5 M€/an...
- Futur aux économies d'énergie
- ⇝ Classement supplémentaire
  - TOP Green500: most powerful supercomputers running the Linpack benchmark ranked by energy efficiency
  - Little Green500: most energy-efficient supercomputers achieving at least 9 tflops on the linpack benchmark
  - Open Green500: exploratory list for energy-efficient supercomputers achieving more than 9 tflops on linpack however they wish
  - HPCC Green500: exploratory lists for most energy-efficient supercomputers running the HPCC benchmark

`http://www.green500.org`

Top Green 500 — 11/2009

Performances en TFLOPS

Rank	*MFLOPS/W*	*Site*	*Computer/Year Vendor*	Power (kW)	TOP500 Rank
1	722.98	Forschungszentrum Juelich (FZJ)	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus	59.49	110
1	722.98	Universitaet Regensburg	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus	59.49	111
1	722.98	Universitaet Wuppertal	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus	59.49	112
4	458.33	DOE/NNSA/LANL	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Infiniband	276	29
4	458.33	IBM Poughkeepsie Benchmarking Center	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Infiniband	138	78
6	444.25	DOE/NNSA/LANL	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband	2345.5	2
7	428.91	National Astronomical Observatory of Japan	GRAPE-DR accelerator Cluster, Infiniband	51.2	445
8	379.24	National SuperComputer Center in Tianjin/NUDT	NUDT TH-1 Cluster, Xeon E5540/E5450, ATI Radeon HD 4870 2, Infiniband	1484.8	5
9	378.77	King Abdullah University of Science and Technology	Blue Gene/P Solution	504	18
9	378.77	EDF R&D	Blue Gene/P Solution	252	49

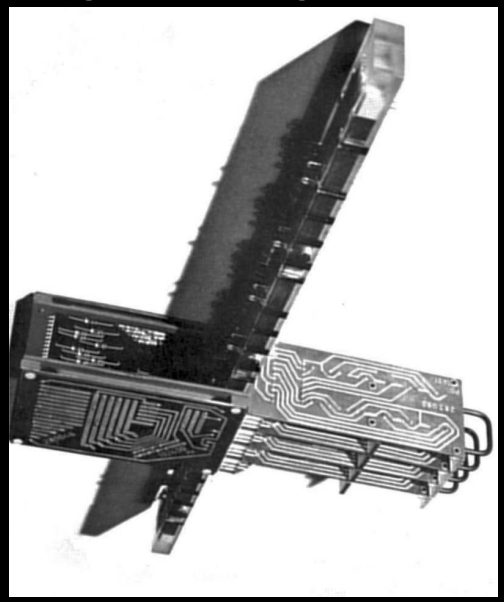
Tendances

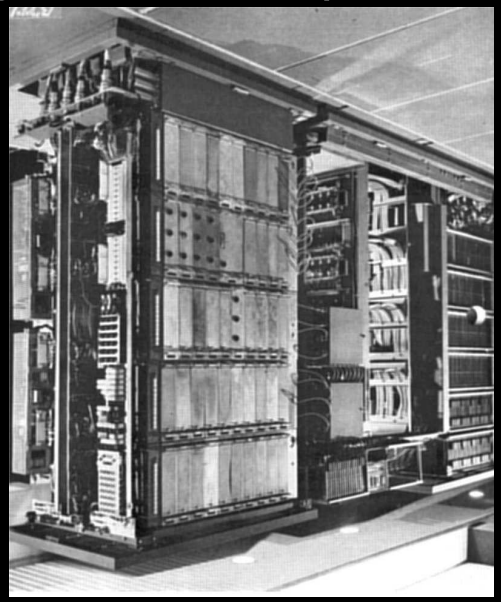
- Fréquence des processeurs : n'évolue plus
- Loi de Moore apporte encore des transistors
  - Toujours plus de parallélisme : *many*cores
  - Architectures hétérogènes : GPGPU, Cell, vectoriel/SIMD, FPGA
- Compilateurs toujours derrière... ☹

Multicores strike back... (I)

Gamma 60 from Compagnie des Machines Bull

- Increase the performance
- 100 kHz memory clock
- *Heterogeneous* multicore because the memory is too... fast! ☺
- 24-bit words, 96 KiB of core memory
- Punch-cards with ECC, magnetic tapes, magnetic drums
- Highly integrated logic in 1-mm germanium bipolar lithography ☺





Multicores strike back... *(II)*

- Gamma 60 multithread programming with `SIMU` (≈ fork) & `CUT` (launch a program on a functional unit) instructions
- Synchronization barrier by concurrent branching on a same target
- Scheduling of threads based on a queue per functional unit stored just... inside the code after each `CUT` instruction!
- Optional hardware critical section on subprograms (cf. `synchronized` of Java)

Multicores strike back... *(III)*

- Installation around 1959
- Already hard to program since the concepts was not here, at most the (grand-)parents of anyone who were to know them...

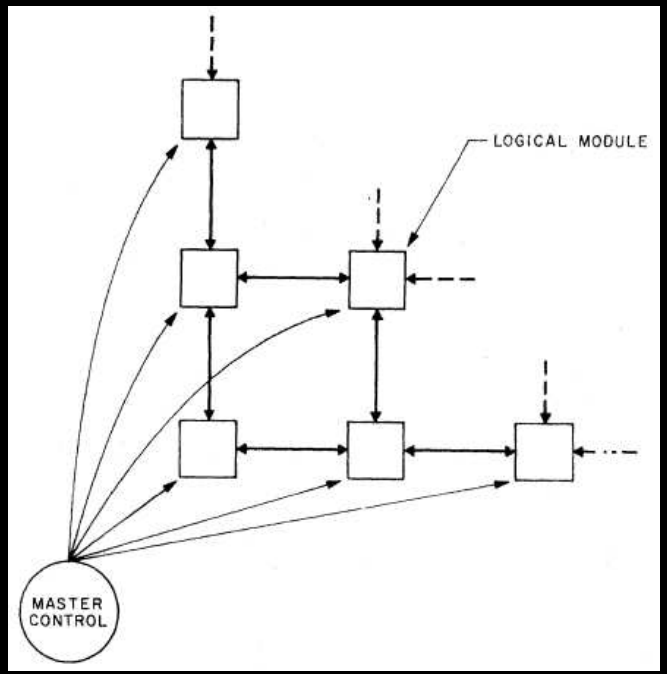
`http://www.feb-patrimoine.com/projet/gamma60/gamma_60.htm`

GPGPUs: just more integrated... (I)

- The “Distributed Computer”
  - Toward computing on spatial data : pattern recognition, mathematical morphology...
  - Massive parallelism to reduce the cost
  - SIMD

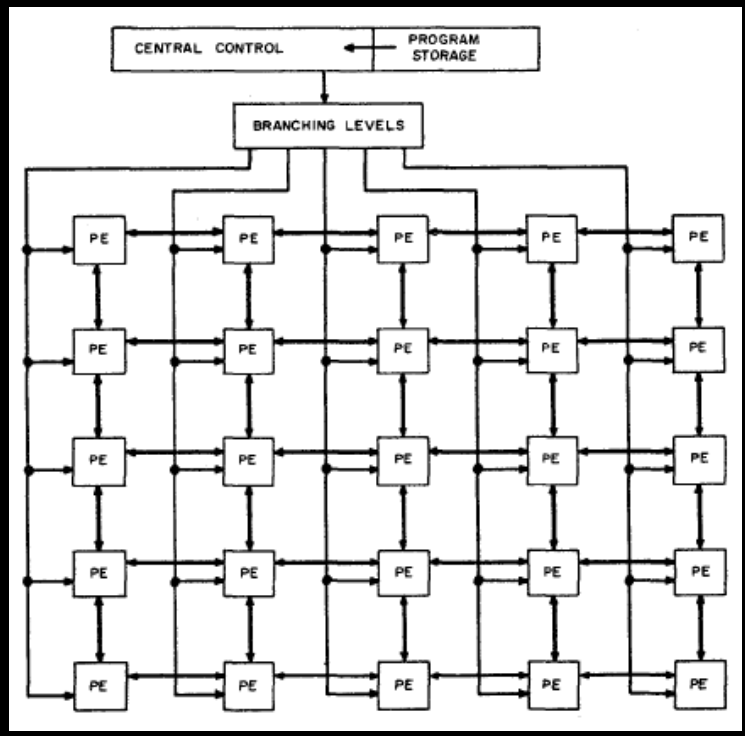
  S. H. Unger. Â« A Computer Oriented Toward Spatial Problems. Â» *Proceedings of the IRE*. p. 1744–1750.

oct. 1958



GPGPUs: just more integrated... *(II)*

- SOLOMON
  - Target application: “data reduction, communication, character recognition, optimization, guidance and control, orbit calculations, hydrodynamics, heat flow, diffusion, radar data processing, and numerical weather forecasting”
  - Diode + transistor logic in 10-pin TO5 package



Daniel L. Slotnick. Â« The SOLOMON computer. Â» *Proceedings of the December 4-6, 1962, fall joint computer conference*. p. 97–107. 1962

Outline

Sictel

- Side effect of Microtel hackerspace in Limoges: Minitel servers on Goupil 3 with 6 modems
- 1986: Minitel servers with clusters of Atari 1040ST (128 users/Atari!) against ATT 3B15
- MIDI 32 kb/s LAN ☺, PC+X25 cards as front-end
- 50% of the total French 3614 chat at Minitel climax
- In a basement of Maison-Alfort

HyperParallel Technologies (1992–1998)

- Parallel computer
- Proprietary 3D-torus network
- DEC Alpha 21064 + FPGA
- HyperC (follow-up of PompC @ LI/ENS Ulm)
  - PGAS (Partitioned Global Address Space) language
  - An ancestor of UPC...
- Already on the Saclay Plateau ! ☺

Quite simple business model

- Customers need just to rewrite all their code in HyperC ☺
- Difficult entry cost... ☹

- Niche market... ☹
- American subsidiary with dataparallel datamining application acquired by Yahoo! in 1998
- Closed technology ⇝ lost for customers and... founders ☹

HyperParallel Technologies (1992–1998)

- Parallel computer
- Proprietary 3D-torus network
- DEC Alpha 21064 + FPGA
- HyperC (follow-up of PompC @ LI/ENS Ulm)
  - PGAS (Partitioned Global Address Space) language
  - An ancestor of UPC...
- Already on the Saclay Plateau ! ☺

Quite simple business model

- Customers need just to rewrite all their code in HyperC ☺
- Difficult entry cost... ☹

- Niche market... ☹
- American subsidiary with dataparallel datamining application acquired by Yahoo! in 1998
- Closed technology ⇝ lost for customers and... founders ☹

Present motivations: reinterpreting Moore's law *(I)*

The good news ☺

- Number of transistors still increasing
- Memory storage increasing (DRAM, FLASH...)
- Hard disk storage increasing
- Processors (with captors) everywhere
- Network is increasing

- The bad news ☹
  - Transistors are so small they leak... Static consumption
  - Superscalar and cache are less efficient compared to transistor budget
  - Storing and moving information is expensive, computing is cheap: change in algorithms...
  - Light's speed as not improved for a while... Hard to reduce latency
    - Chips are too big to be globally synchronous at multi GHz ☹
  - pJ and physics become very fashionable

Present motivations: reinterpreting Moore's law *(II)*

- Power efficiency in $\mathcal{O}(\frac{1}{f})$
  - Transistors cannot be used at full speed without melting ☹
  - Heat
- I/O and pin counts
  - Huge time and energy cost to move information outside the chip ☹

- Rotating hard disk with 1D density d increase
  - Storage in $\mathcal{O}(d^2)$
  - But track speed only $\mathcal{O}(d)$

  ⇝ Reading all the disk in $\mathcal{O}(\frac{1}{d})$ ☹

⇝ 2006: Time to be back in parallelism!

Yet another start-up... ☺

Heterogeneous parallelism

Parallelism is the only way to go...

- Research is just crossing reality!
- Scaring... ☹
- Exciting! ☺

No one size fit all...

Future will be heterogeneous

The “Software Crisis”

Edsger DIJKSTRA, 1972 Turing Award Lecture, « The Humble Programmer »

“To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.”

`http://en.wikipedia.org/wiki/Software_crisis`

But... it was before parallelism democratization! ☹

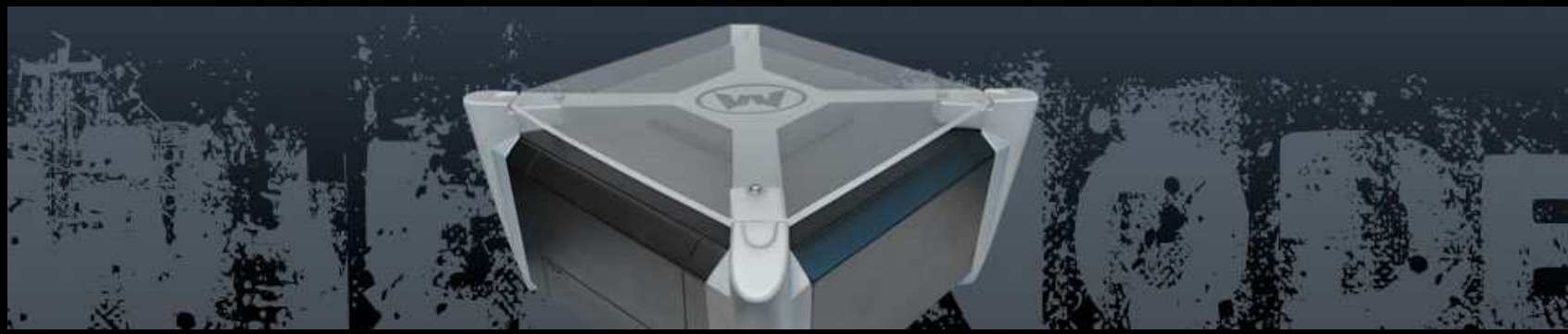
Time to be back in parallelism!

- Good time for more start-ups! ☺
- ⇝ 2006: thinking to yet another start-up...
- People that met ≈ 1990 at the French Parallel Computing military lab SEH/ETCA
- Later became researchers in Computer Science, CINES director and ex-CEA/DAM, venture capital and more: ex-CEO of Thales Computer, HP marketing...
- HPC Project launched in December 2007
- ≈ 30+ colleagues in France (Montpellier, Meudon), Canada (Montréal with Parallel Geometry) & USA (Santa Clara, CA)

HPC Project hardware: WildNode from Wild Systems

Through its Wild Systems subsidiary company

- WildNode hardware desktop accelerator
  - Low noise for in-office operation
  - *x*86 manycore
  - nVidia Tesla GPU Computing
  - Linux & Windows



- WildHive
  - Aggregate 2-4 nodes with 2 possible memory views
    - Distributed memory with Ethernet or InfiniBand

`http://www.wild-systems.com`

HPC Project software and services

- Parallelize and optimize customer applications, co-branded as a bundle product in a WildNode (e.g. Presagis Stage battle-field simulator, Wild Cruncher for Scilab//...)
- Acceleration software for the WildNode
  - CPU+GPU-accelerated libraries for C/Fortran/Scilab/Matlab/Octave/R
  - Automatic parallelization for Scilab, C, Fortran...
  - Transparent execution on the WildNode
- Par4All automatic parallelization tool
- Remote display software for Windows on the WildNode

HPC consulting

- Optimization and parallelization of applications
- *High Performance?...* not only TOP500-class systems: power-efficiency, embedded systems, green computing...
- ⇝ Embedded system and application design
- Training in parallel programming (OpenMP, MPI, TBB, CUDA, OpenCL...)

Outline

Évolution logicielle

Jusqu'à présent...

- Langages d'assemblage
- Langages de haut niveau pour machines à la von NEUMANN (Fortran, C...)
- Programmation orientée objet pour composabilité, malléabilité et maintenabilité de gros programmes
- Bibliothèques de composants, outils, patrons de conception, spécifications, modélisation, méthodologies, tests...

Hautes performance ?

Bah... Loi de MOORE ☺

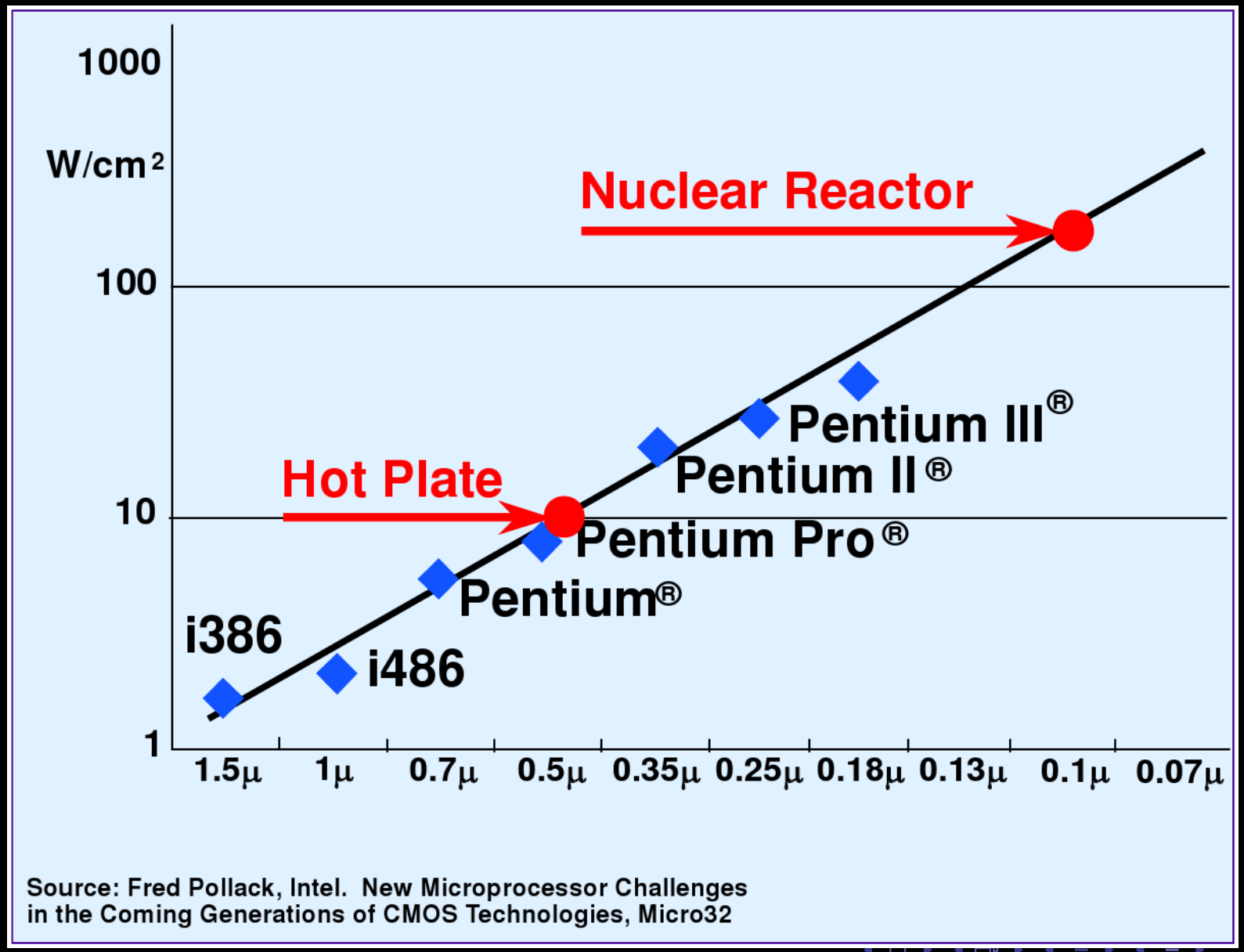
Programmeurs inconscients des processeurs...

- On ne mélange pas les ~~mouchoirs en dentelle~~softeux avec les ~~serpillières~~hardeux ☹
- Langages de haut niveau loin du matériel (encore pire avec JVM, CLI...)
- Abstraction qui a permis beaucoup de liberté créatrice aux programmeurs ☺
- L'avenir est au Web 3.0 ! ☺
- La vitesse ? Travailleurs de l'ombre (loi de MOORE...) font qu'un programme antique va beaucoup plus vite aujourd'hui sur n'importe quel processeur ! ☺

⇝ Un programmeur peut tout ignorer des processeurs

$\exists^?$ encore cours d'architecture des ordinateurs en école d'ingénieurs ? ☹

Densité de puissance



Fin de l'augmentation des performances séquentielle

- Passe d'un facteur 2 tous les 1,5 ans à tous les ≈5 ans... ☹
- Pourtant besoin de toujours plus de performances
  - Devise de Delphes « rien de trop » μηδὲν ἄγαν
  - Données traitées plus grandes
  - Plus de fonctionnalités par €
  - Plus de fonctionnalités par W

Seule solution : faire du parallélisme...

...et garder le moral : « composabilité, malléabilité et maintenabilité, portabilité... »

Comment faire face ?

Exemple projet logiciel dans monde selon Moore

Déroulement

- 2010 : démarrage du projet logiciel : 8 cœurs par circuit intégré
- fin 2011 : sortie première version : 16 cœurs par circuit intégré
- 2013 : seconde version, 32 cœurs par circuit intégré

Serez-vous prêt(e) ?

Dans le monde des écoles d'ingénieur...

- 2010 : un élève intègre. Programmation séquentielle
- 2012 : un élève en stage. Ouch ! $\exists$ parallélisme
- 2013 : un élève dans la vraie vie... Faire du parallélisme sans formation
- 2016 : un élève passe sa thèse, 256 cœurs/circuit... 

Heureusement la formation continue existe... ☺

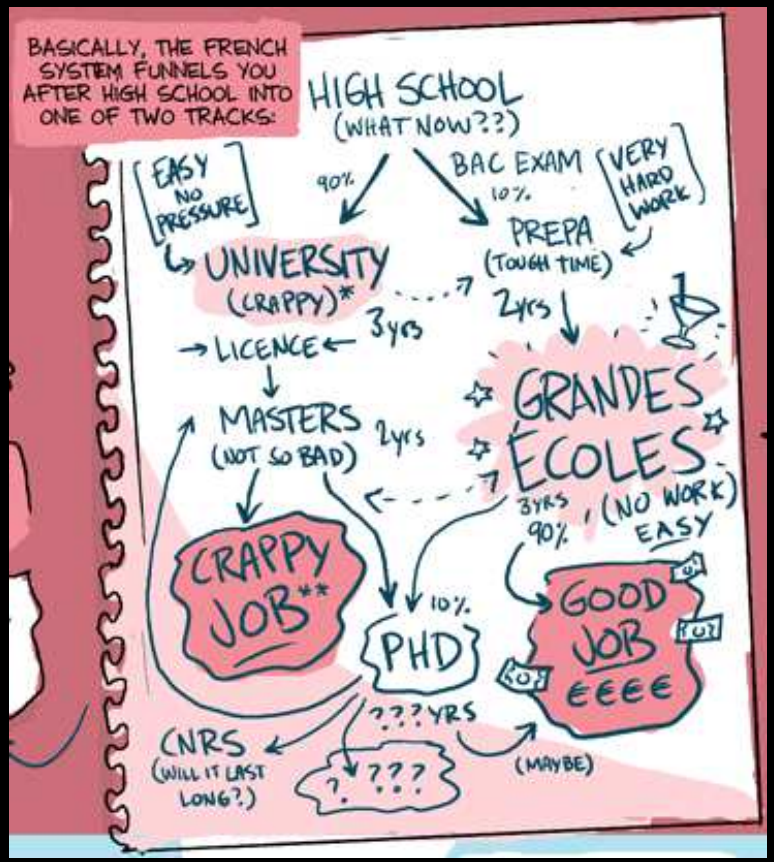
Dure réalité du parallélisme (I)

http://www.phdcomics.com/comics/archive.php?comicid=1292

- Rajoute du sucre syntaxique
- Problème de DRH
- Importance croissante des pannes...

PhDcomics du 3/15/2010, visite de Jorge CHAM à l'X

Dure réalité du parallélisme *(II)*



We *really* need an elevator...

Outline

Outline

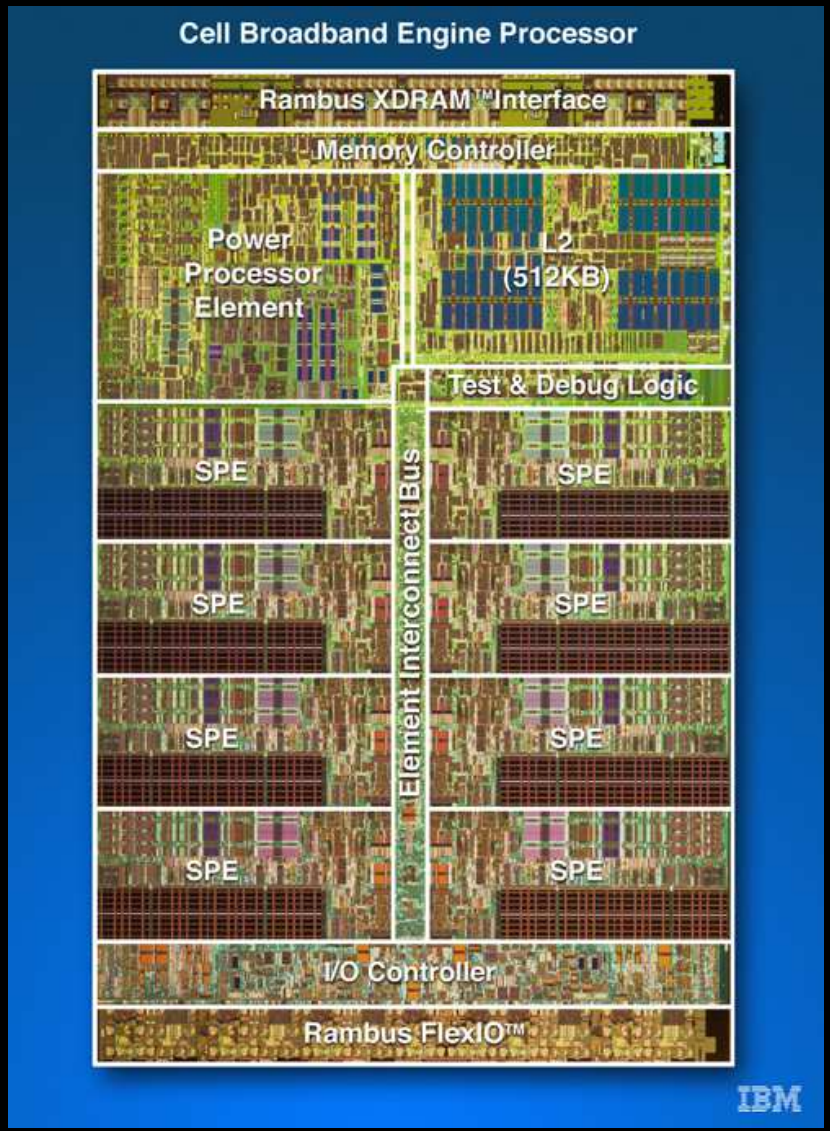
Sony PS/3

- 256 Mio XDR Rambus (et 256 Mio de mémoire vidéo GDDR3)
- Processeur MP-SoC Cell multi-cœur
- Processeur graphique nVidia RSX ≈NV47 400 GFLOPS
- Lecteur Blu-ray et prise HDMI
- Disque dur de 60 Go
- Ethernet 1 Gb/s & WiFi
- GNU/Linux (`/proc`...)
- Achetée à TÉLÉCOM Bretagne/HPCAS ☺

- 400 € en 2008

http://fr.wikipedia.org/wiki/PlayStati

Processeur STI Cell



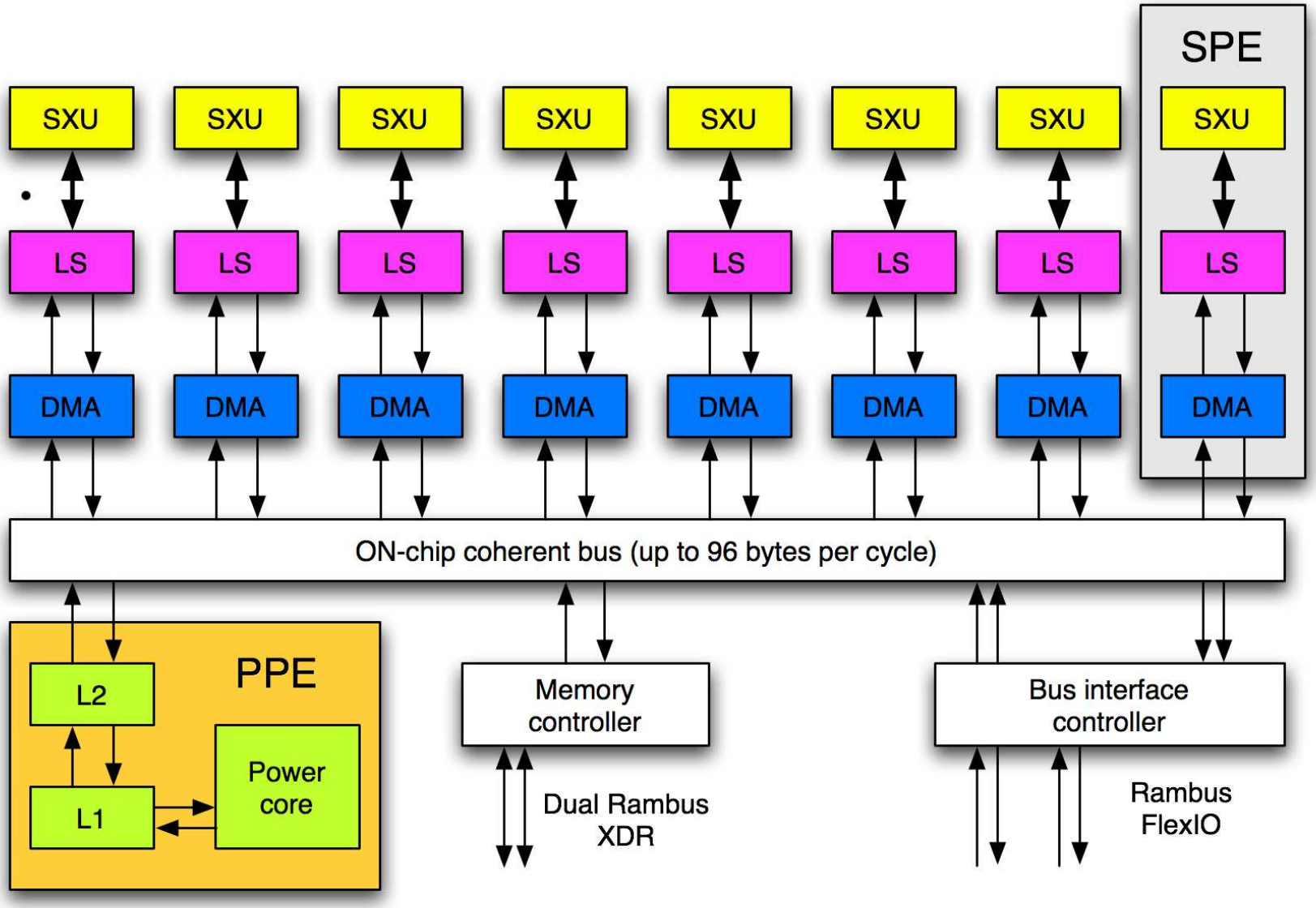
(*Multi-Processor-System On Chip*) 3,2 GHz, 90 puis 65 nm, multi-cœur

- 1 PowerPC SMT 2 voies, 512 Kio L2
- 8 SPE PowerPC SIMD
- 256 Kio mémoire locale/SPE
- Débit bus mémoire 25,6 Gio/s
- 204,8 GFLOPS SP, 3,3 Tbops
- NoC 307 Go/s
- Opérateurs DMA et synchronisation

Base aussi de la RoadRunner, qui a passé le PFLOPS en 2008

MP-SoC

Architecture du STI Cell



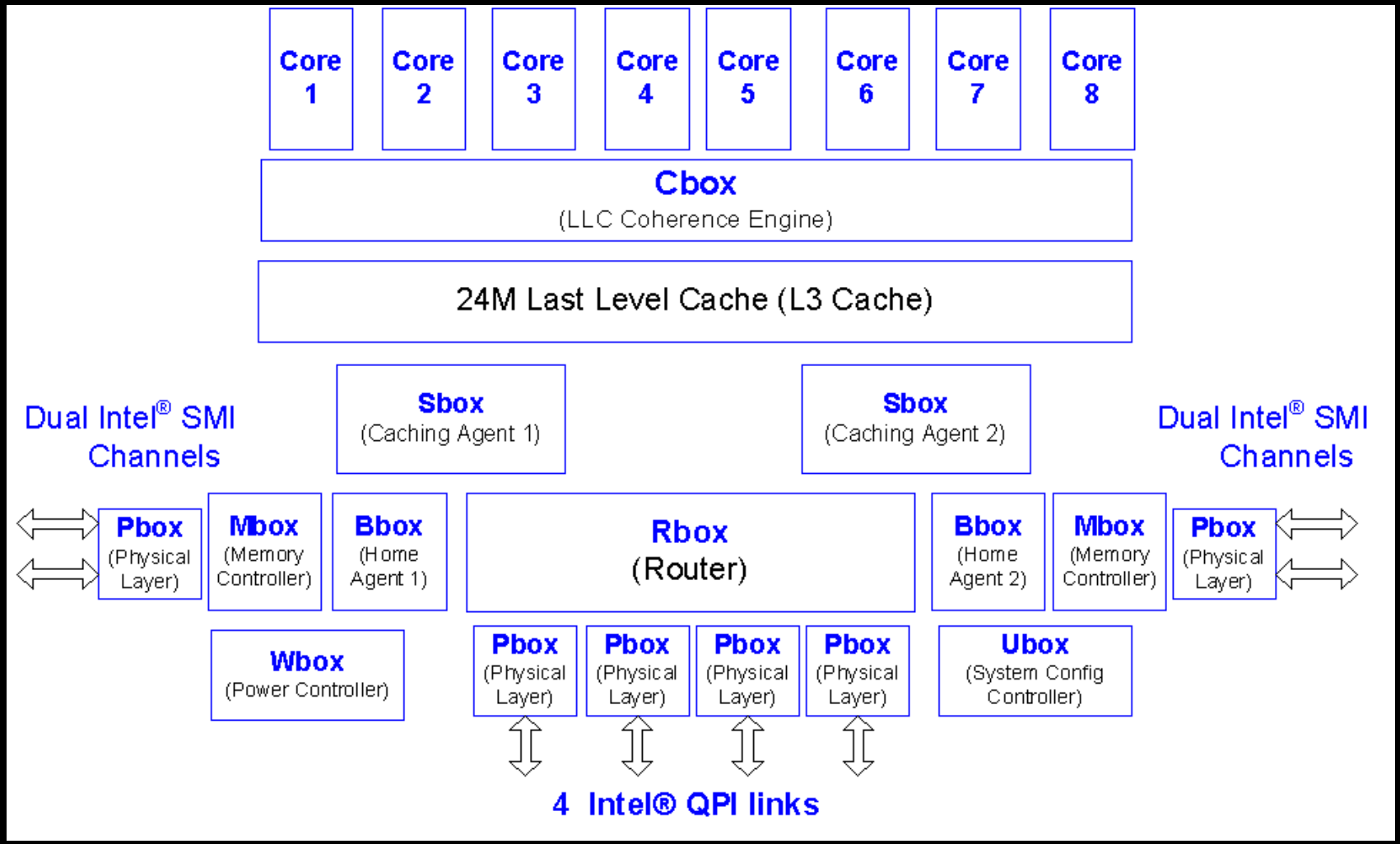
Intel Nehalem (I)

- x86-64 new microarchitecture
- With on-chip memory controllers like AMD Opteron: ↗ bandwidth, ↘ latency
- SSE4.2 instruction set: strings (XML parser...), comparisons (data-mining), CRC & cryptography (protocols)
- Power consumption fine tuning with many sensors (power, temperature). Possible to speed up when less cores are used (turbo)
- Xeon X7560 (Beckton microarchitecture), 2010
  - 8 cores @ 2.27 GHz (+ turbo 2.666 GHz) + SMT 2 threads (HyperThreading) with 256 KB L2 cache/core, 32D+32I KB cache/core
  - 24 MB L3 cache
  - 4 Quick Path Interconnects (QPI) @ 6.4 GT/s to play Lego with processors & accelerators (≈HyperTransport d'AMD)

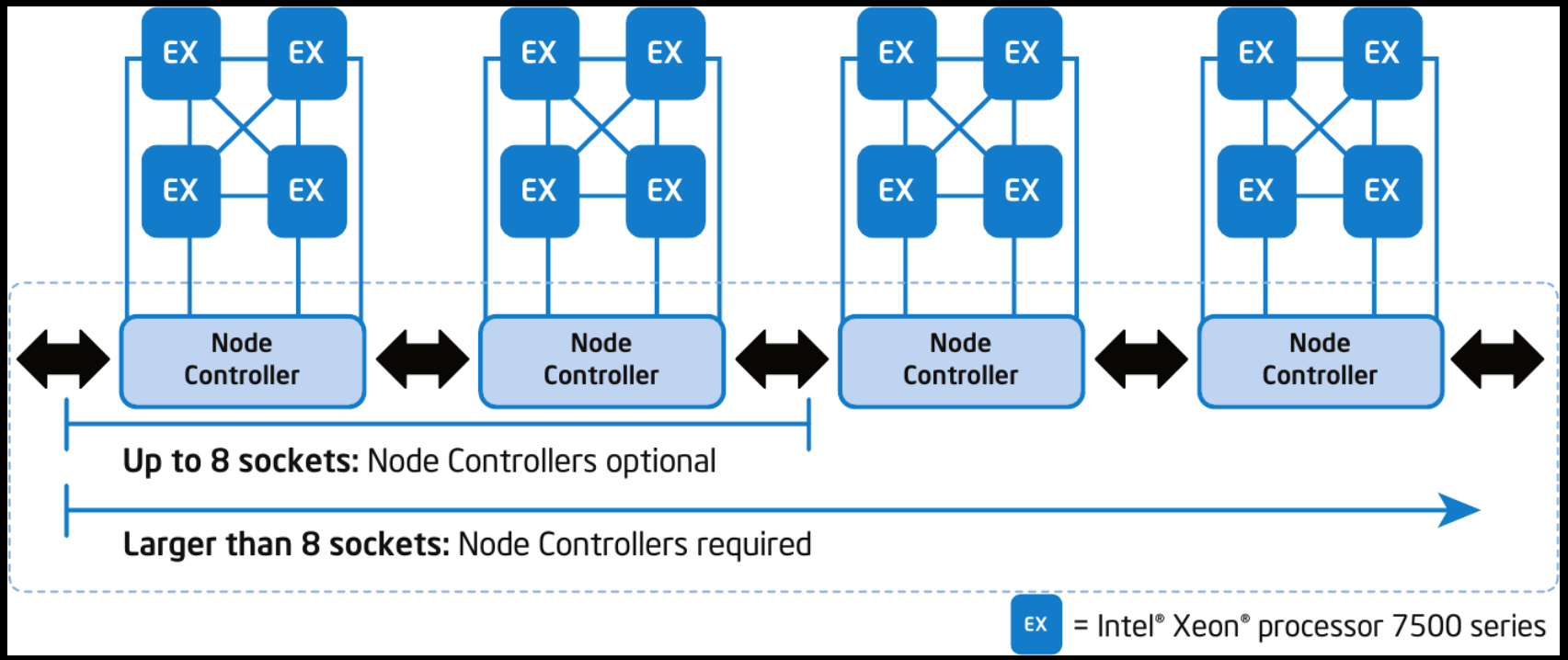
Intel Nehalem (II)

- 4 DDR3-1333 MHz memory controllers
- 130 W
- $3692 March 30, 2011
- 2.3 Gtr 45 nm 684 mm^2

Intel Nehalem *(III)*



Intel Nehalem (IV)

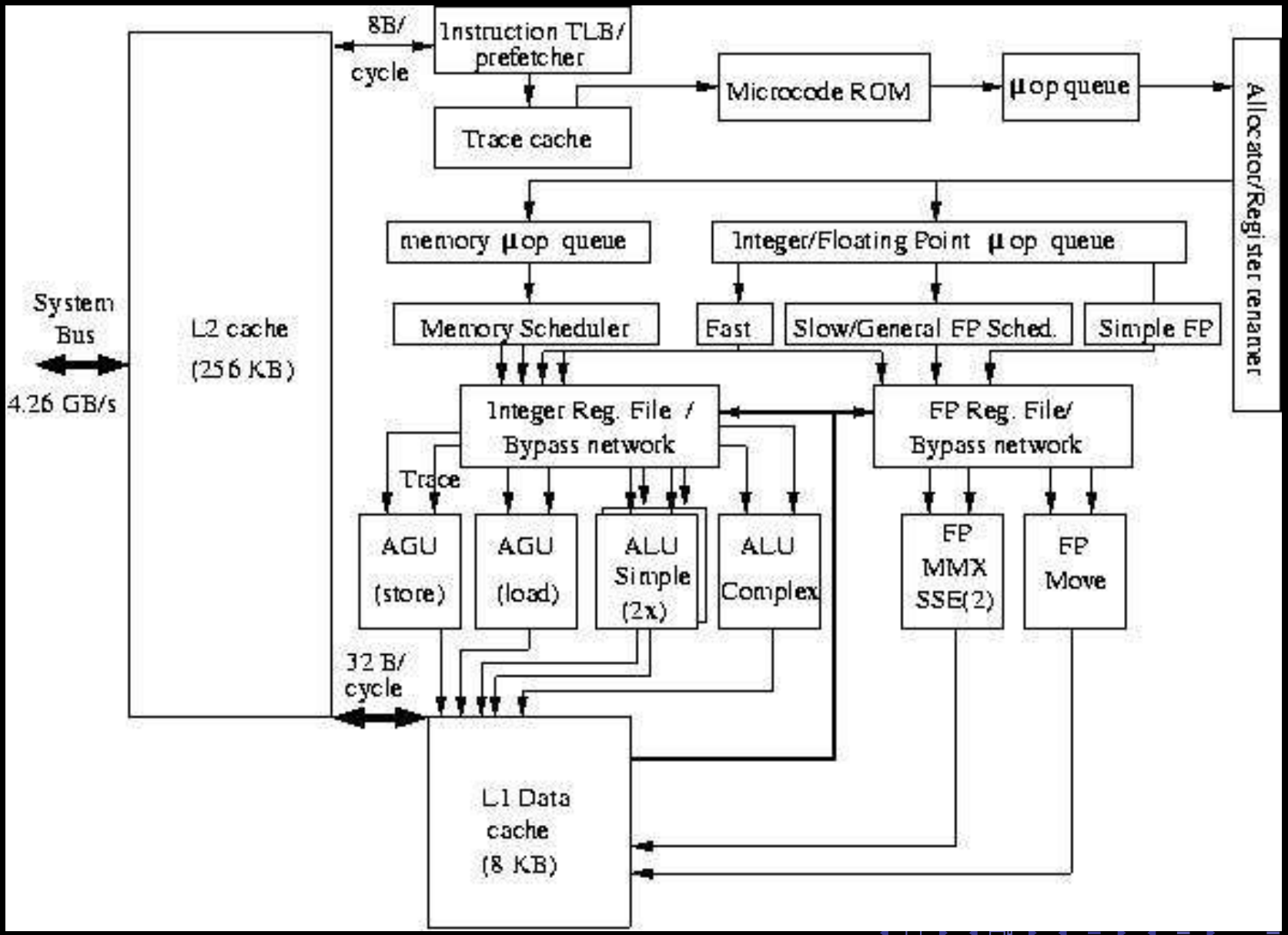


Processeur superscalaire (I)

Dépasser une instruction par cycle en lançant plusieurs instructions par cycle [Tjaden and Flynn, 1970, Riseman and Foster, 1972] :

- Matériel complexe pour préserver la sémantique (dépendances) $\Longrightarrow$ tableau noir, algorithme de TOMASULO,...
- Exploitation du parallélisme « naturel » du code :
  - Compatibilité logicielle
  - Peu d'efforts au compilateur (théorie...)
- Exploitation du code réarrangé [Foster and Riseman, 1972] et déroulé pour performances maximales

Intel Pentium 4

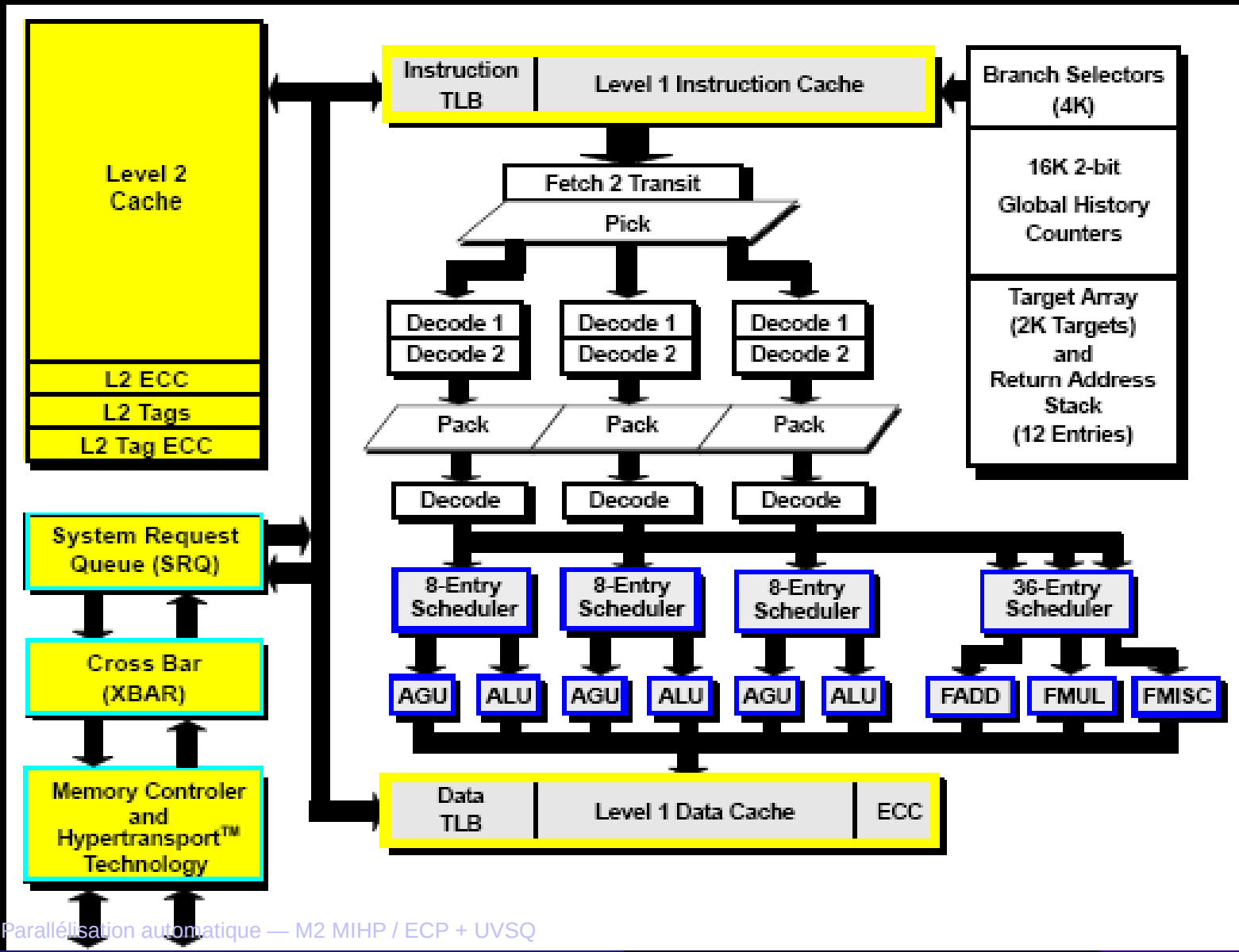


AMD Opteron 6180 (2011) (I)

- *x*86-compatible instruction set
- 64-bit mode that doubles also register numbers
- Out-of-order superscalar execution with 9 instructions/cycle
  - 3 integer instructions
  - 3 address generators
  - 3 floating computation operators (+, *, memory)
- Opteron 6180 SE, 02/2011, $1514
  - 12 cores @ 2.5 GHz, 512 KB/core L2 cache
  - 2 × 6 MB L3 cache
  - 21 GB/s DDR3 memory
  - 4 x 6.4 GT/s HyperTransport
  - 1.8 Gtr 45 nm 692 mm^2
  - 140 W

AMD Opteron 6180 (2011) *(II)*

http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/



IBM Power 7 (2010)

- 4 chips per quad-chip module
- 8 cores per chip @ 4.0 GHz (4.25 GHz in TurboCore mode with 4 cores)
- 1.2 Gtr 45 nm SOI process, 567 mm^2
- 4 SMT threads per core
- 32I+32D kB L1 cache/core
- 256 kB L2 cache/core
- 32 MB L3 cache in eDRAM
- 100 GB/s DDR3 memory
- 12 execution units per core:
  - 2 fixed-point units
  - 2 load/store units
  - 4 double-precision floating-point units
  - 1 vector unit supporting VSX
  - 1 decimal floating-point unit
  - 1 branch unit
  - 1 condition register unit

Cartes graphiques (I)

- Graphismes de plus en plus compliqués ⇝ besoin de vrais calculs
- Deviennent des ordinateurs parallèles
- GPGPU ≡ *General-purpose computing on graphics processing units*

`http://www.anandtech.com/video/showdoc.aspx?i=3341`

`http://www.anandtech.com/video/showdoc.aspx?i=3408`

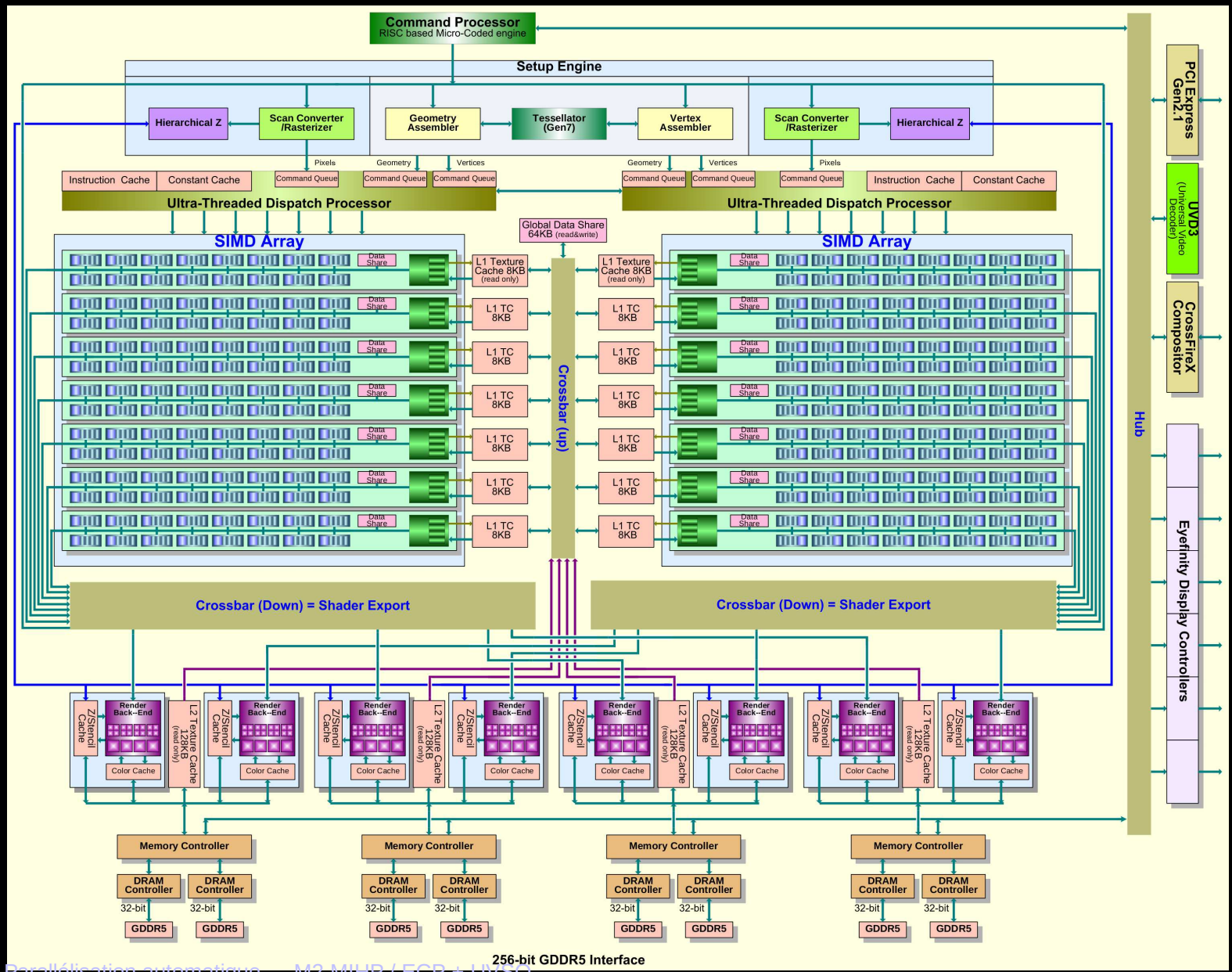
Off-the-shelf AMD/ATI Radeon HD 6970 GPU

- 2.64 billion 40nm transistors
- 1536 stream processors @ 880 MHz, 2.7 TFLOPS SP, 675 GFLOPS DP
- + External 1 GB GDDR5 memory 5.5 Gt/s, 176 GB/s, 384b GDDR5
- 250 W on board (20 idle), PCI Express 2.1 x16 bus interface
- OpenGL, OpenCL
- ∃ Radeon HD 6990 double chip card

More integration:

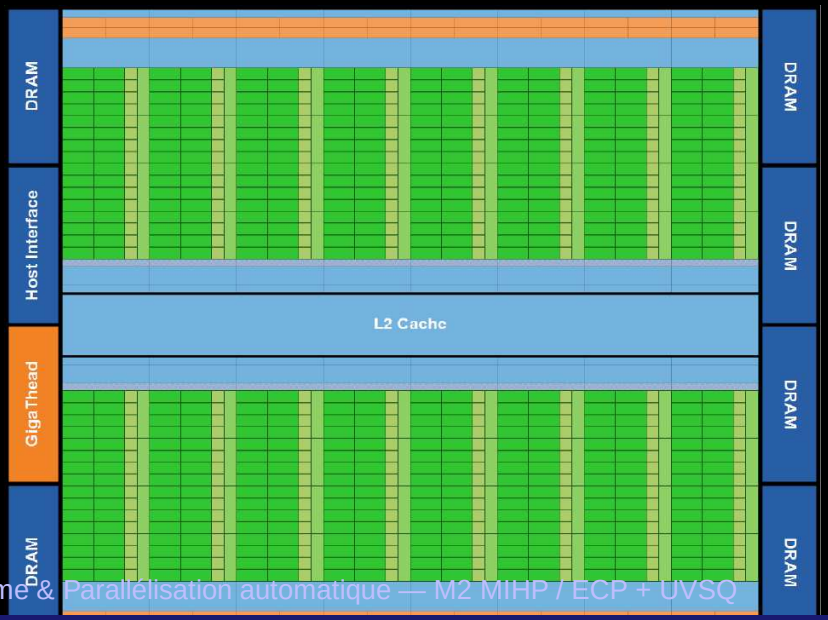
- Llano APU (FUSION Accelerated Processing Unit) : x86 multicore + GPU 32nm, OpenCL

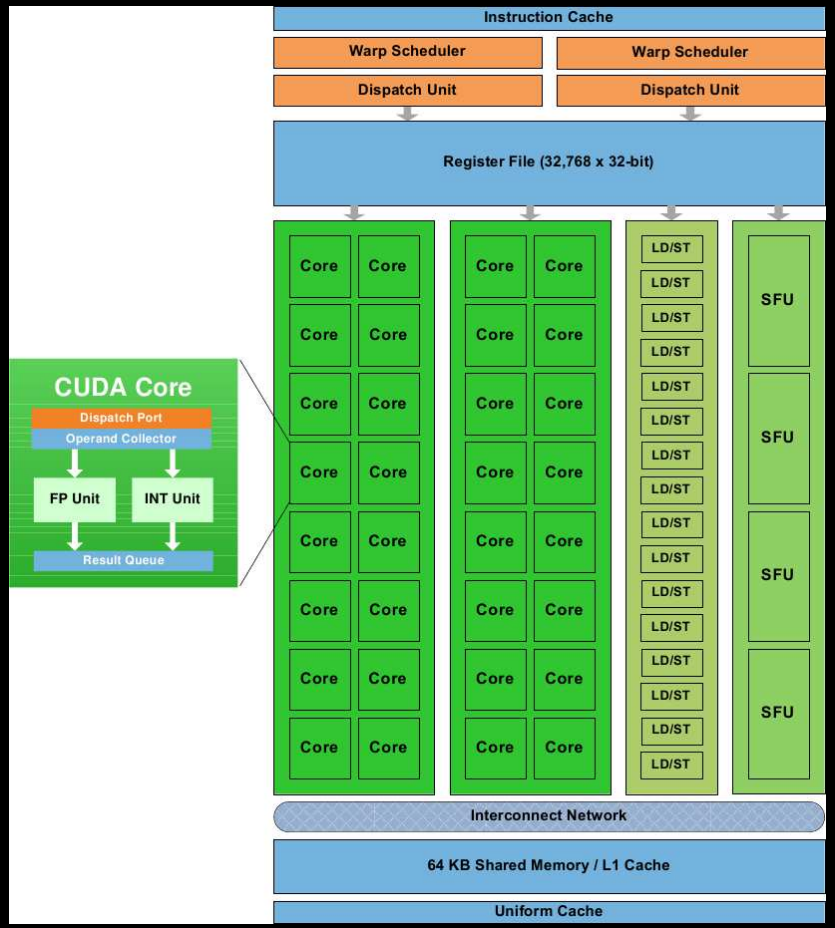
Radeon HD 6870 — big picture



Off-the-shelf nVidia Tesla Fermi M2090 & GTX580

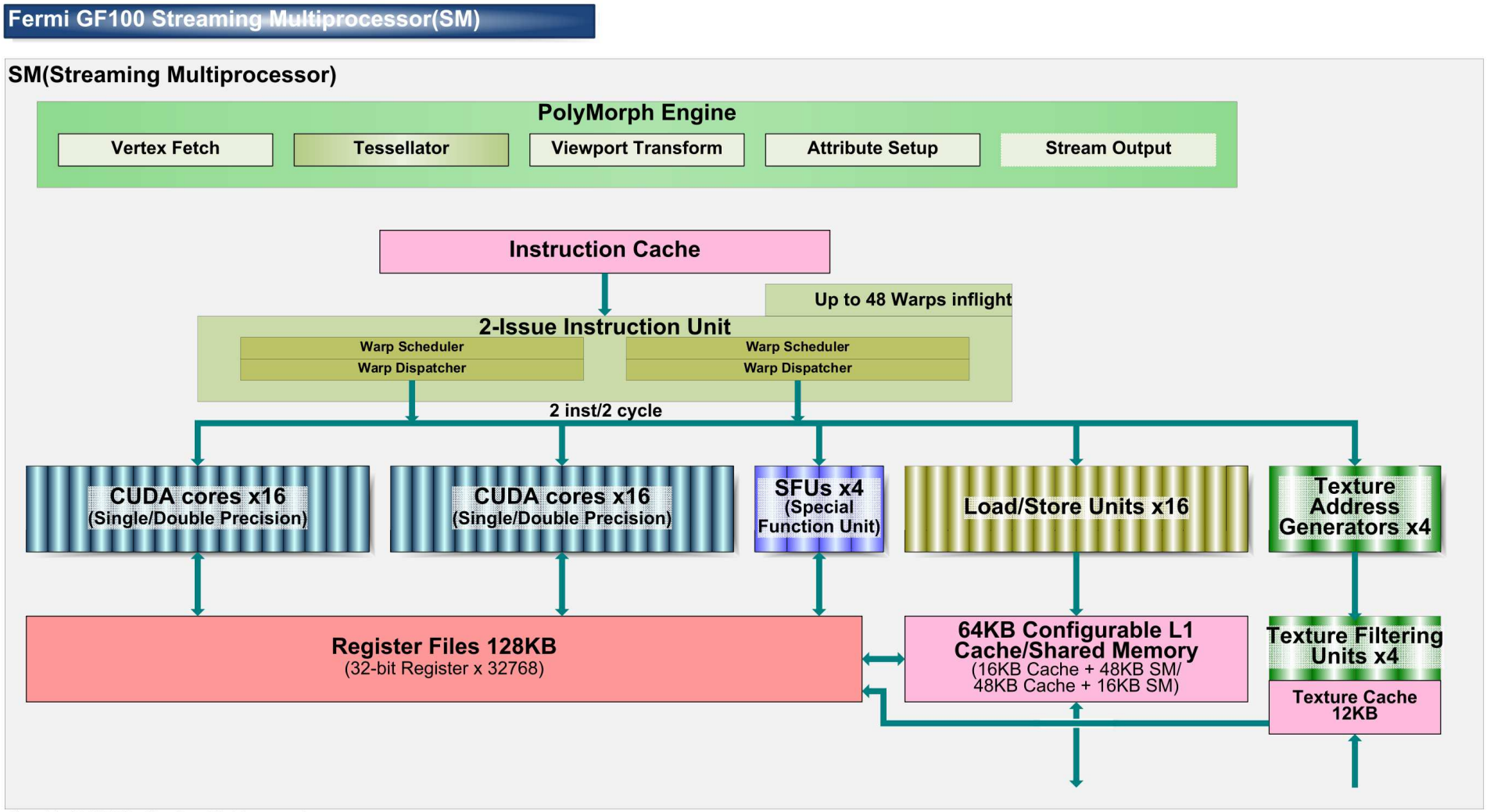
- GF110: 3 billion 40nm tr.
- 512 thread processors @ 1300 MHz, 1,3 TFLOPS SP, 666 GFLOPS DP
- + External 6 GB GDDR5 ECC memory 3,7 Gt/s, 177 GB/s. Less if using ECC





- 247 W on board PCI Express 2.1 x16 bus interface
- OpenGL, OpenCL, CUDA
- GTX580 double chip card

GF100 Stream Multiprocessor

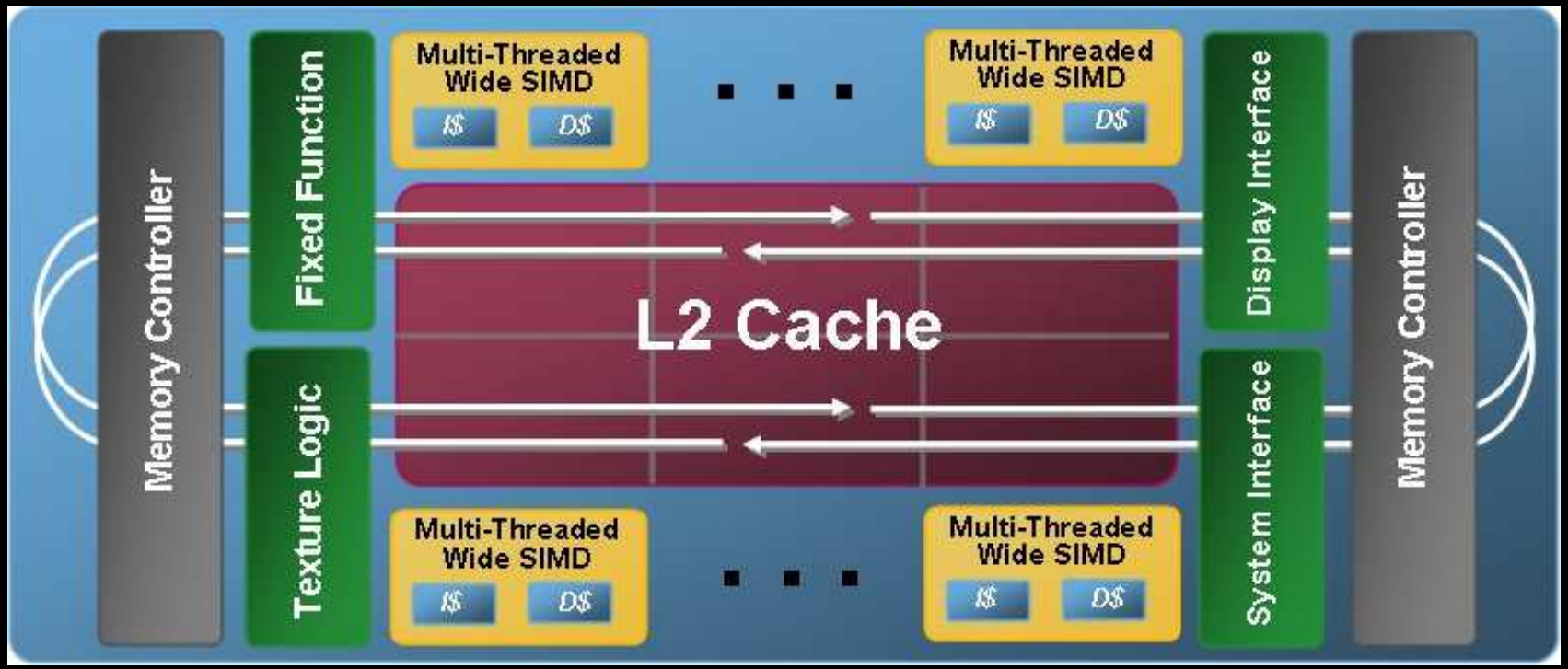


Évolution processeurs Intel

- Processeurs de plus en plus complexes sur ligne principale
  - Multicœurs
  - Superscalaires agressifs
  - Gravure 45 nm en 2008
- Pas de GPU haut de gamme
- Pas adapté pour de la puissance de calcul intensive

Intel MIC — Concepts de Larrabee

- Mettre plus de processeurs plus simples
- Partir d'un vieux Pentium 0,8 μm
- On peut en loger 10+ sur un circuit avec techno moderne
- Superscalaire dans l'ordre
- Amélioration des opérations graphiques



Comparaison Core 2 Duo & Intel MIC

Comparaison	Core 2 Duo	Intel MIC
# of CPU Cores	2 out of order	32 in-order
Instructions per Issue	4 per clock	2 per clock
VPU Lanes per Core	4-wide SSE	16-wide
L2 Cache Size	4MB	4MB
Single-Stream Throughput	4 per clock	2 per clock
Vector Throughput	8 per clock	**512** per clock

1 TFLOPS @ 1 GHz with FMA/MAC

`http://anandtech.com/cpuchipsets/intel/showdoc.aspx?i=3367`

Malédiction d'Intel ? (I)

Jusqu'à présent, tout essai d'Intel pour faire original a été un échec commercial...

- iAPX 432
- i860
- Itanium
- Larrabee gelé... revient avec MIC bientôt

Toujours du *x*86... ☹

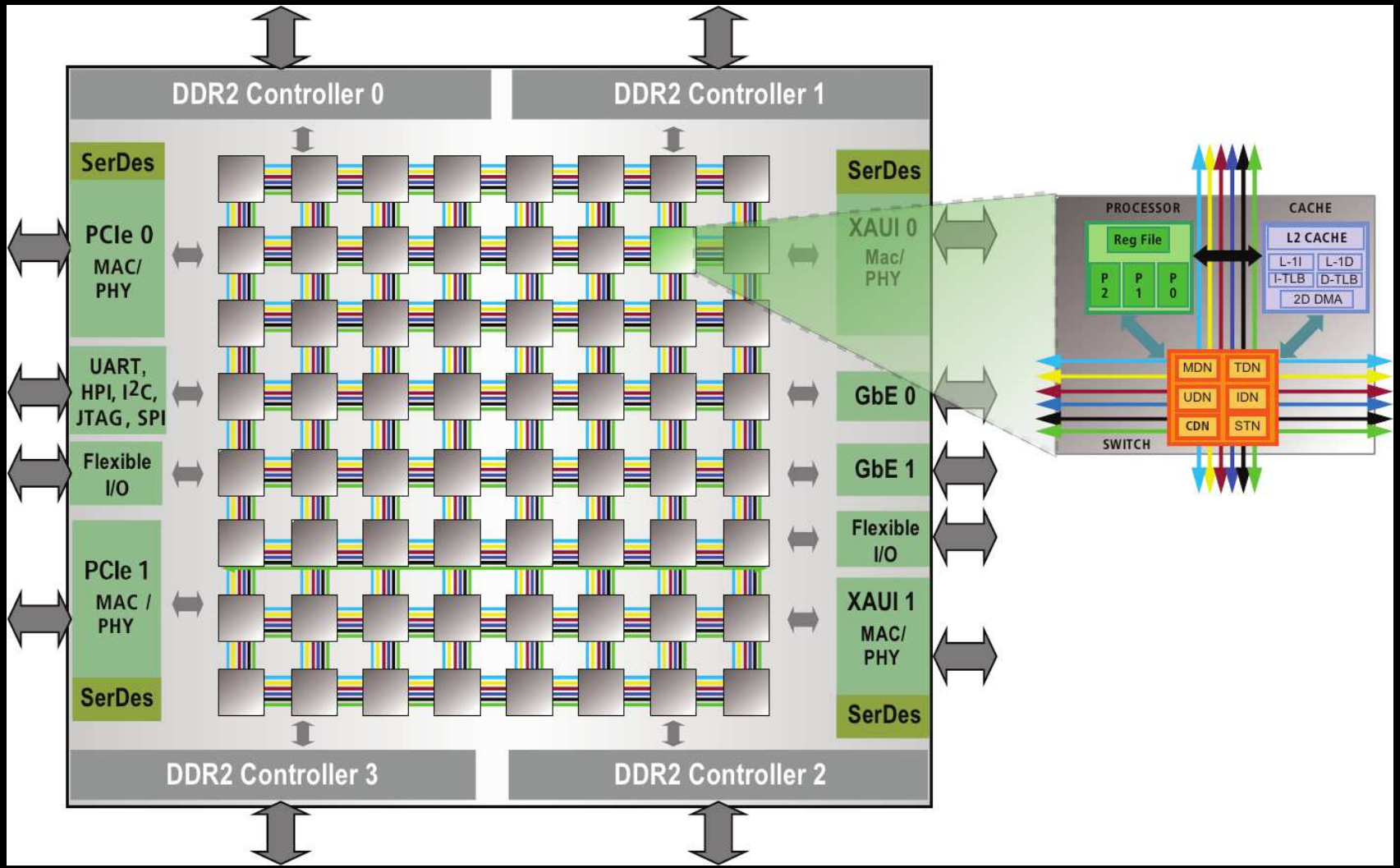
ARM yourself (I)

- Do some computations where the captors are...
- Smartphone and other sensor networks
- Trade-off between communication energy and inside/remote computations
- Texas Instrument OMAP4470 announced on 2011/06/02
  - 2 ARM Cortex-A9 MPCores @ 1.8GHz with Neon vector instructions
  - 2 ARM Cortex-M3 cores (low-power and real-time responsiveness, multimedia, avoiding to wake up the Cortex-A9...)
  - **SGX544 graphics core with OpenCL 1.1 support**, with 4 USSE2 core @ 384 MHz producing each 4 FMAD/cycle: 12.3 GFLOPS
  - 2D graphics accelerator
  - 3 HD displays and up to QXGA (2048x1536) resolution + stereoscopic 3D
  - Dual-channel, 466 MHz LPDDR2 memory

ARM yourself *(II)*

- ⇝ Current course to have non-*x*86 servers based on ARM...
- ∃ Experiments on low power clusters
- Think to evaluate power consumption on your application

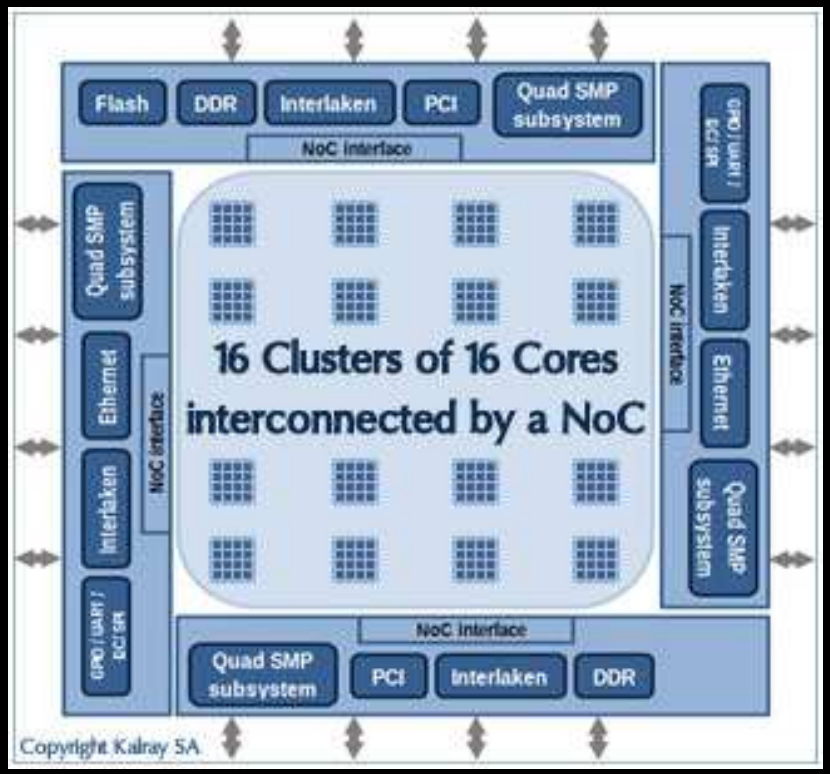
Tilera TilePro64 (I)



Tilera TilePro64 *(II)*

- Processing power with good network interfaces
  - Interactive data analysis
  - Video/audio codec
  - DPI, IDS, IPsec...
- 8x8 processors with SMP, partitionable
- 32-bit VLIW with SIMD mode
- 700-866 MHz: 443 GOPS 8 bits, 23 W
- 64 DDR2 controllers 25.6 GB/s
- 2× 10 GbE XAUI + IP session hash distribution
- SMP Linux or bare bone per tile
- C/C++ gcc compiler
- TMC library for hardcore support (2D network...)
- Eclipse support with graphical simulator
- OpenMP

MPPA de Kalray (the French touch!) (I)



- 256 VLIW processors per chip 16 clusters of 16 processors)
- Shared memory and NoC with DMA
- 28 nm CMOS technology, $\approx$ 5 W @ 400 MHz
- FPU 32/64 bits IEEE 754: 205 GFLOPS SP, 1 TOPS 16 bits
- 2$\times$ 64-bit DDR3 memory controllers for high bandwidth main memory transfers
- 2$\times$ 40 Gb/s or 8$\times$ 10 Gb/s) Ethernet controller

MPPA de Kalray (the French touch!) *(II)*

- 2× 8-lane PCI Express Gen 3
- 4× 4–8-lane Interlaken interfaces for multi-MPPA chip system integration (8 MPPA/PCIe board) or connection to external FPGAs, I/O...
- Linux or bare metal with AccessCore library
- Multi-core compiler (gcc 4.5), simulator, debugger (gdb), profiler
- Eclipse IDE
- Programming from high-level C-based language
- AccessCore library

FPGA (I)

- Field-programmable gate array with bitstream configuration in memory
- Xilinx Virtex 7
  - 2M logic cells (6-LUT), 28 nm
  - 85Mb block RAM
  - 5280 DSP slices (6.7 TFMA/s)
  - 96 transceiver @ 28Gb/s: 2.8 Tb/s
  - PCIe gen3 $\times 8$
  - 1200 pins
- Radar, communications, HPC, datamining, bioinformatics...

FPGA (II)

- VHDL-to-bitstream compiler... but **hard work**
- Dynamic partial reconfiguration
- ∃ C-to-VHDL compilers
  - Riverside Optimizing Compiler for Configurable Computing (ROCCC): open source
  - Impulse C
  - Catapult C
  - Cynthesizer
  - ...

Convey HC-1^{ex}

- Intel Xeon quad-core @2.13 GHz with 128 GB
- 4 Xilinx Virtex 6 LX760 FPGA with 128 GB DDR2
- 1520 W in 3U rack
- Linux
- Various instruction sets (“personality”)
- Fortran/C/C++ vectorizing compiler replacing complex instructions by FPGA vectorized implementation
- Possible to design its own instruction set (ROCCC...)
- 401$\times$ speed-up on Smith-Waterman algorithm compared to $x86$

Anton computer from D. E. Shaw

Success story...

1. Create a hedge fund
2. Earn a lot of money
3. Spend it by creating a 500+ people start-up in bioinformatics
4. Build from scratch (ASIC) a computer for solving special issues in computational biology and chemistry

PacketShader

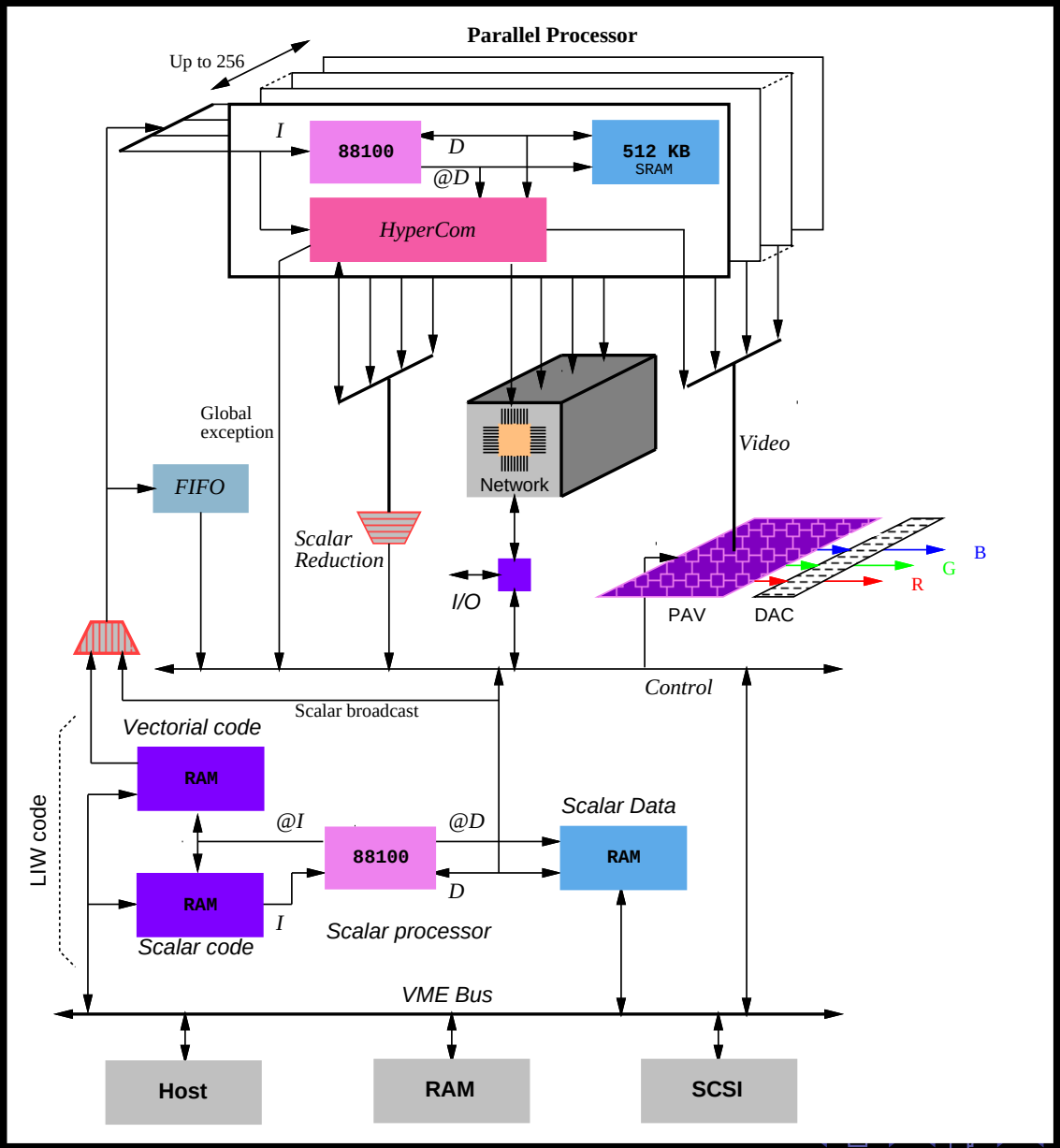
- Use Linux PC + GPU as a router with processing power
  - 2 4-core Nehalem @2.66 GHz + 4 10 GigE NIC + 2 GPU GTX480
- 40 Gb/s routing even with 64-byte packets
- 8–20 Gb/s IPsec tunneling
- Linux IP stack to slow ⇝ own raw device driver for Intel 82598/82599 NIC
  - Extract Ethernet flow into **user** mode
  - Huge recycled packet buffers
  - Batch processing to amortize overhead
  - NUMA-aware processing: packets processed by NIC-local CPU in its RAM, aligned in cache
- Current bottleneck: IOH chipset

`http://shader.kaist.edu/packetshader`

OpenFlow

- Need for advanced routing with computing capabilities
- PC or Tilera-like with few 10+ Gb/s Ethernet or Infiniband: bandwidth maybe OK but not enough links
- ⇝ Open standard to interact with existing routers & switch: OpenFlow `http://www.openflow.org`
- Possible to extract *minimal* flows to feed PC/GPU/MP-SoC/FPGA accelerators and reinject results in the router
- OpenFlow implemented by many router companies
- Possible with non-OpenFlow but less flexible and non portable

POMP & PompC @ LI/ENS 1987–1992



Present motivations

- MOORE's law there are more transistors but they cannot be used at full speed without melting ☹
- Superscalar and cache are less efficient compared to transistor budget
- Chips are too big to be globally synchronous at multi GHz ☹
- Now what cost is to move data and instructions between internal modules, not the computation!
- Huge time and energy cost to move information outside the chip

Parallelism is the only way to go...

Research is just crossing reality!

No one size fit all...

Future will be heterogeneous: GPGPU, Cell, vector/SIMD, FPGA, PIM...

But compilers are always behind... ☹

Outline

Outline

La nouvelle donne du renouveau informatique *(I)*

« The Landscape of Parallel Computing Research: A View from Berkeley », Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams and Katherine A. Yelick. EECS Department University of California, Berkeley Technical Report UCB/EECS-2006-183, December 18, 2006

`http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf`

Révision des bonnes vieilles hypothèses (*conventional wisdoms*)...

1. Old CW: Power is free, but transistors are expensive.
   New CW is the “Power wall”: Power is expensive, but transistors are “free”. That is, we can put more transistors on a chip than we have the power to turn on.

La nouvelle donne du renouveau informatique *(II)*

2. Old CW: If you worry about power, the only concern is dynamic power.
   New CW: For desktops and servers, static power due to leakage can be 40% of total power.
3. Old CW: Monolithic uniprocessors in silicon are reliable internally, with errors occurring only at the pins.
   New CW: As chips drop below 65 nm feature sizes, they will have high soft and hard error rates. [Borkar 2005] [Mukherjee et al 2005]
4. Old CW: By building upon prior successes, we can continue to raise the level of abstraction and hence the size of hardware designs.
   New CW: Wire delay, noise, cross coupling (capacitive and inductive), manufacturing variability, reliability (see above), clock jitter, design validation, and so on conspire to stretch the

La nouvelle donne du renouveau informatique *(III)*

development time and cost of large designs at 65 nm or smaller feature sizes.

5. Old CW: Researchers demonstrate new architecture ideas by building chips.
   New CW: The cost of masks at 65 nm feature size, the cost of Electronic Computer Aided Design software to design such chips, and the cost of design for GHz clock rates means researchers can no longer build believable prototypes. Thus, an alternative approach to evaluating architectures must be developed.
6. Old CW: Performance improvements yield both lower latency and higher bandwidth.
   New CW: Across many technologies, bandwidth improves by at least the square of the improvement in latency. [Patterson 2004]

La nouvelle donne du renouveau informatique *(IV)*

7. Old CW: Multiply is slow, but load and store is fast.
   New CW is the “Memory wall” [Wulf and McKee 1995]: Load and store is slow, but multiply is fast. Modern microprocessors can take 200 clocks to access Dynamic Random Access Memory (DRAM), but even floating-point multiplies may take only four clock cycles.
8. Old CW: We can reveal more instruction-level parallelism (ILP) via compilers and architecture innovation. Examples from the past include branch prediction, out-of-order execution, speculation, and Very Long Instruction Word systems.
   New CW is the “ILP wall”: There are diminishing returns on finding more ILP. [Hennessy and Patterson 2007]

La nouvelle donne du renouveau informatique *(V)*

9. Old CW: Uniprocessor performance doubles every 18 months.
   New CW is Power Wall + Memory Wall + ILP Wall = Brick Wall.
   Figure 2 plots processor performance for almost 30 years. In 2006, performance is a factor of three below the traditional doubling every 18 months that we enjoyed between 1986 and 2002. The doubling of uniprocessor performance may now take 5 years.
10. Old CW: Don't bother parallelizing your application, as you can just wait a little while and run it on a much faster sequential computer.
    New CW: It will be a very long wait for a faster sequential computer (see above).
11. Old CW: Increasing clock frequency is the primary method of improving processor performance.
    New CW: Increasing parallelism is the primary method of improving processor performance.

La nouvelle donne du renouveau informatique *(VI)*

12 Old CW: Less than linear scaling for a multiprocessor application is failure.
New CW: Given the switch to parallel computing, any speedup via parallelism is a success.

Grain du parallélisme

Taille moyenne des tâches élémentaires (en nombre d'instruction, taille mémoire, temps)
Gros grain $\rightarrow$ grain fin :

- Programmes
- Procédures & fonctions
- Instructions
- Expressions
- Opérateurs
- Bits

Choix en relation avec l'architecture cible

Degré du parallélisme

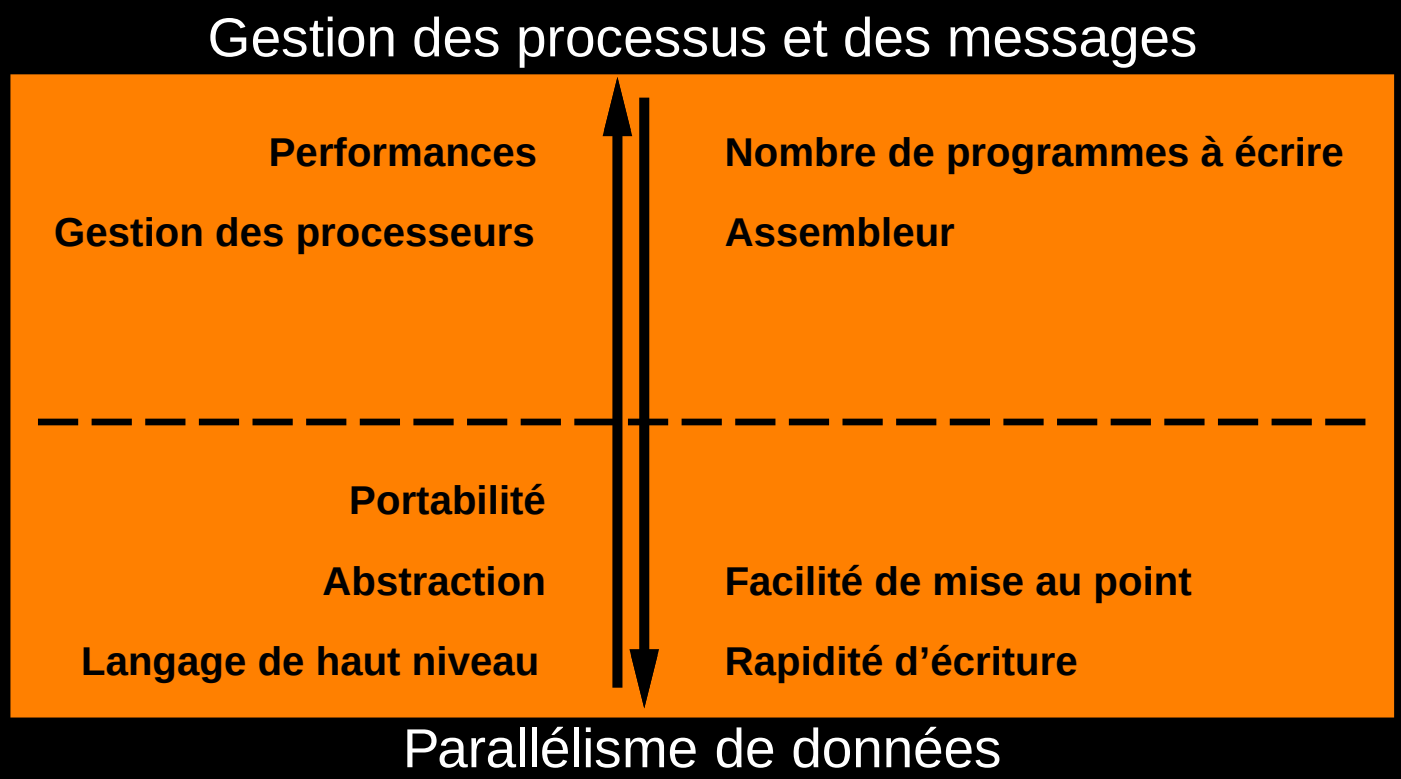
- Nombre d'opérations exécutées simultanément
- Idée du nombre de processeurs nécessaires
- Si moins de processeurs, repliement du parallélisme
- Peut varier au cours de l'exécution
- Si trop de processeurs : inactivité ↗

Programmation des machines parallèles

Principalement 2 niveaux :

- Gestion totale du parallélisme « à la main » au niveau des processeurs
- Langage de haut niveau abstrayant le parallélisme de la machine

Compromis à trouver...



Programmation ex(a)tensible

- ⚠ Ne pas essayer de programmer avec *quelques* threads/processus/données parallèles
- Il faut des *centaines* ou des *milliers* de threads !
  - Paralléliser code : souvent compliqué
  - Prend du temps
  - ⚠ Sera en retard par rapport aux nouvelles machines sinon... ☹
- Penser « pannes »
  - Déploiement correction d'erreur à tout niveau
    - Correction d'erreur 1 bit/mot classique dans ordinateurs professionnels, devrait arriver dans GPGPU
    - *Check-pointing* : comment sauver 200 To à 200 Go/s (disque SAS coûte 10× SATA2 ☹) ? 20 mn sur IBM RoadRunner, 30 mn sur IBM BG/P IDRIS...
    - Avec 10^6 cœurs en 2012-2013 : ≈ 1 panne/heure
    - ⇝ Domaine de recherche en explosion (INRIA Futur LRI...)
  - Surtout pannes logicielles en fait (heureusement ?)
  - Algorithmique tolérante aux pannes ? Oui dans certains cas spécifiques (solveurs itératifs)

⇝ Programmation ex(a)tensible pour viser machines exaflopiques à venir

Outline

Extracting parallelism in applications...

- The implicit attitude
  - Hardware: massively superscalars processors
  - Software: auto-parallelizing compilers
- The ($\pm$) explicit attitude
  - Languages ($\pm$ extensions): OpenMP, UPC, HPF, Co-array Fortran (F- -), Fortran 2008, X10, Chapel, Fortress, Matlab, SciLab, Octave, Mapple, LabView, nVidia CUDA, AMD/ATI Stream (Brook+, Cal), OpenCL, HMPP, *insert your own preferred language here* ...
  - Framework: MapReduce, Hadoop...
  - Libraries: application-oriented (mathematics, coupling...), parallelism (MPI, concurrency *pthreads*, SPE/MFC on Cell...), Multicore Association API (MCAPI...), objects (parallel STL, TBB, Ct...)

... but multidimensional heterogeneity!

Welcome into Parallel Hard-Core Real Life on Chip 2.0!

- Heterogeneous execution models
  - Multicore ($\pm$S)MP $\pm$ coupled by caches
  - SIMD instructions in processors (Neon, VMX, SSE 4.2, LRBni...)
  - Hardware accelerators (MIMD, MISD, SIMD, SIMT, FPGA, ASIC...)
- New heterogeneous memory hierarchies
  - Classic caches/physical memory/disks
  - Flash SSD is a new-comer to play with
  - NUMA (*Non Uniform Memory Access*): sockets-attached memory banks, remote nodes...
  - Peripherals attached to sockets: NUPA (*Non Uniform Peripheral Access*). GPU on PCIe $\times$16 in this case...
  - If non-shared memory: remote memory, remote disks...
  - Inside GPU: registers, local memory, shared memory, constant memory, texture cache, processor grouping, locked physical pages, host memory access...
- Heterogeneous communications
  - Anisotropic networks
  - Various protocols

Several dimensions to cope with at the same time ☹

Dwarfs d'applications parallèles

- `http://view.eecs.berkeley.edu/` : « The Landscape of Parallel Computing Research: A View From Berkeley »
- Essaye de capturer des exemples typiques courants qui peuvent servir à analyser et concevoir des architectures

	Dwarf	Performance Limit: Memory Bandwidth, Memory Latency, or Computation?
1	Dense Matrix	Computationally limited
2	Sparse Matrix	Currently 50% computation, 50% memory BW
3	Spectral (FFT)	Memory latency limited
4	N-Body	Computationally limited
5	Structured Grid	Currently more memory bandwidth limited
6	Unstructured Grid	Memory latency limited
7	MapReduce	Problem dependent
8	Combinational Logic	CRC problems BW; crypto problems computationally limited
9	Graph traversal	Memory latency limited
10	Dynamic Programming	Memory latency limited
11	Backtrack and Branch+Bound	?
12	Construct Graphical Models	?
13	Finite State Machine	Nothing helps!

Extraire du parallélisme (I)

Exemple de calcul de polynômes de vecteurs (livre « Initiation au parallélisme, concepts, architectures et algorithmes », Marc GENGLER, Stéphane UBÉDA & Frédéric DESPREZ)

pour $i = 0$ à $n - 1$ faire
    $vv[i] = a + b.v[i] + c.v[i]^2 + d.v[i]^3 + e.v[i]^4 + f.v[i]^5 + g.v[i]^6$
fin pour

Calcul avec parallélisme de donnée (typique SIMD) ≡ faire en parallèle la même chose sur des données différentes :

pour $i = 0$ à $n - 1$ faire en parallèle
    $vv[i] = a + b.v[i] + c.v[i]^2 + d.v[i]^3 + e.v[i]^4 + f.v[i]^5 + g.v[i]^6$
fin pour

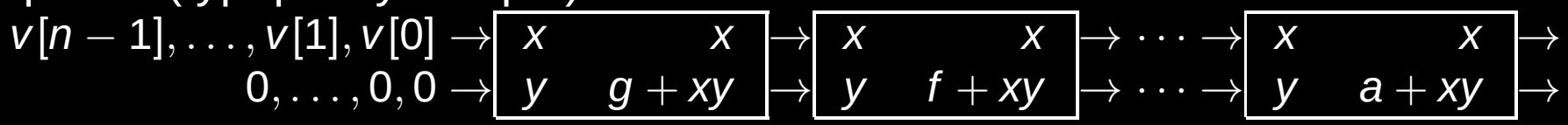
Extraire du parallélisme *(II)*

Découpage en tâches (typique MIMD) ≡ faire des choses différentes sur des données différentes :

```
pour i = 0 à n − 1 faire
    tâches parallèles
        x = a + b.v[i] + c.v[i]^2 + d.v[i]^3
    ||
        y = e. + f.v[i] + g.v[i]^2
    ||
        z = v[i]^4
    fin tâches parallèles
    vv[i] = x + z.y
fin pour
```

Extraire du parallélisme *(III)*

Pipeline (typique systolique) $\equiv$ travail à la chaîne :



7 étages de pipeline (7 processeurs) traitant 1 flux de plusieurs données.

Type de parallélisme (I)

Parallélisme de données :

- Régularité des données
- Même calcul à des données distinctes

Parallélisme de contrôle :

- Fait des choses différentes

Parallélisme de flux : pipeline

- Régularité des données
- Chaque donnée subit séquence de traitements

Réingénierie pour le parallélisme

« Reengineering for Parallelism: An Entry Point into PLPP (Pattern Language for Parallel Programming) for Legacy Applications », Berna L. Massingill, Timothy G. Mattson, Beverly A. Sanders

`http://www.cise.ufl.edu/research/ParallelPatterns/plop2005.pdf`

- Guide de recettes pratiques lorsqu'on part d'un vieux programme...
- ... ou qu'on trouve que cela aide de concevoir d'abord un programme séquentiel (mais  ...)
- 4 espaces de conception à traverser en partant du problème, contexte et des utilisateurs
  - « Trouver de la concurrence » (*Finding Concurrency*)
  - « Structure algorithmique » (*Algorithm Structure*)
  - « Structure de support » (*Supporting Structures*)
  - « Mécanismes d'implémentation » (*Implementation Mechanisms*)

Espace de conception « trouver concurrence » (I)

- Permet de structurer problème pour exposer concurrence exploitable
- Travail niveau algorithmique de haut niveau pour exposer concurrence potentielle
- Quelques patrons de conceptions possibles dans cet espace :
  - Patrons de décomposition de problèmes en morceaux concurrents
    - Décomposition en tâches : comment un problème peut-il être décomposé en tâches qui s'exécutent de manière concurrente ?
    - Décomposition des données : comment décomposer données du problème en unités qui peuvent être traitée relativement indépendamment ?

Espace de conception « trouver concurrence » *(II)*

- Patrons d'analyse de dépendances : regroupent tâches et analyse leur dépendances
  - Groupe des tâches : comment regrouper les tâches d'un problème pour simplifier la gestion de leur dépendances ?
  - Ordonnancement des tâches : comment ordonner des groupes de tâches (provenant d'une décomposition d'un problème et d'un regroupement des tâches) pour satisfaire les inter-dépendances ?
  - Partage des données : Comment à partir d'une décomposition des données et en tâches partager des données entre tâches ?
- Évaluation de la conception : est-ce que les résultats de la phase de décomposition et d'analyse de dépendances est suffisamment bonne pour passer à l'espace de conception suivant (structure algorithmique) ou est-ce qu'on réitère conception dans cet espace ?

Espace de conception « structure algorithmique »

(I)

- Restructuration des algorithmes pour exploiter la concurrence potentielle obtenue dans l'espace précédent
- Patrons possibles de stratégies pour exploiter la concurrence :
  - Patrons pour applications centrées sur une organisation en tâche
    - Parallélisme de tâche : comment organiser un algorithme en une collection de tâches à exécution concurrente ?
    - Diviser pour régner : comment exploiter concurrence potentielle dans le cas d'un problème formulé avec stratégie « diviser pour régner » ?
  - Patrons pour applications centrées sur organisation par décomposition de données
    - Décomposition géométrique : comment organiser un algorithme autour de structures de données mises à jour par morceau de manière concurrente ?
    - Données récursives : dans le cas d'opérations sur une structure de donnée récursive (liste, arbre, graphe...) qui semblent séquentielles, comment réaliser ces opérations en parallèle ?

Espace de conception « structure algorithmique »

(II)

- Patrons pour applications orientées flot de données
  - Pipeline : si application peut être vue comme un flux de données à travers une série d'étapes de calcul, comment exploiter cette concurrence ?
  - Coordination par événements : si application décomposée en groupe de tâches semi-indépendantes interagissant de manière irrégulière dépendant des données (donc contraintes de dépendances entre tâches aussi...), comment réaliser cette interaction pour avoir du parallélisme ?

Espace de conception « Structure de support » *(I)*

- Étape intermédiaire entre description algorithmique et implémentation
- Traite de la programmation mais en restant à haut niveau
- Exemples de patrons de conception :
  - Patrons représentant les approches structurant les programmes :
    - SPMD : problèmes liés aux interaction de différentes unités d'exécution. Comment structurer programmes pour gérer au mieux interactions et faciliter intégration dans programme global ?
    - Ferme de travail: Comment organiser un programme conçu avec un besoin de distribuer de manière dynamique du travail à des travailleurs ?
    - Parallélisme de boucles : comment traduire en programme parallèle un programme séquentiel dominé par de gros nids de boucles
    - Fork/Join : Si programme avec nombre de tâches concurrentes qui varient avec relations complexes entre elles, comment construire programme parallèle avec tâches dynamiques ?

Espace de conception « Structure de support » *(II)*

- Patrons représentant des structures de données courantes :
  - Données partagées : comment gérer explicitement des données partagées entre différentes tâches concurrentes ?
  - Files partagées : comment partager de manière correcte une structure de file entre différentes unités d'exécution ?
  - Tableaux distribués. Souvent, tableaux partitionnés sur plusieurs unités d'exécution. Comment faire un programme efficace et... lisible ?
- À ce niveau est aussi discuté d'autres structures comme SIMD, MPMD, client-serveur, langages parallèles déclaratifs, environnements de résolution de problèmes...

Espace de conception « Mécanismes d'implémentatio (I)

- Traite de l'adaptation des espaces de conception de haut niveau à des environnements de programmation particuliers
- Souvent correspondance directe entre choix de cet espace et élément de l'environnement de programmation cible
- Exemple de patrons
  - Gestion des unités d'exécution : parallélisme implique plusieurs entités fonctionnant simultanément qui doivent être gérées (création et destruction de processus lourds ou légers...)
  - Synchronisation : permet de respecter des contraintes d'ordonnancement d'événements sur différences unités d'exécutions (barrière, exclusion mutuelle, barrière mémoire...)
  - Communication : si pas de mémoire partagée, besoin de communications explicites pour échanger des informations entre processus

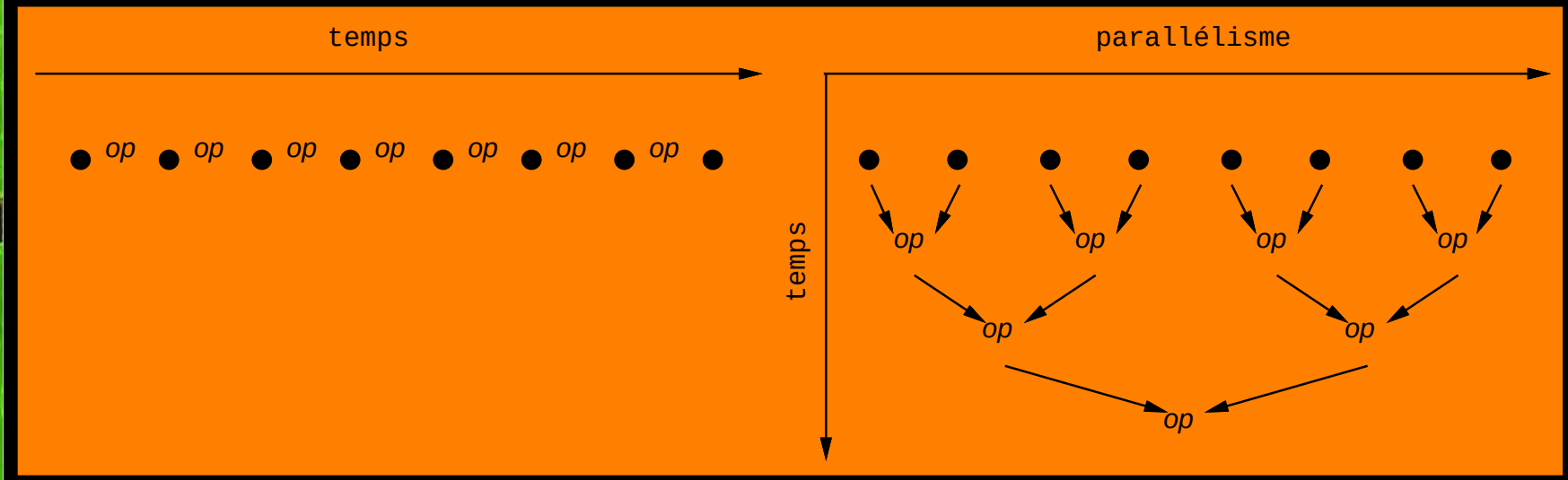
Cycle de développement

1. Analyse de complexité du problème
2. Analyse du programme disponible
   - Analyse performances, profiling (`gprof`, VTune...)
3. Conception de la parallélisation
   - Bibliothèques déjà optimisées (mathématique)
   - Objets parallèles (STL parallèles, TBB...)
   - Langages parallèles (OpenMP, UPC, Fortran 2008...)
   - Bibliothèques parallèles (MPI, threads systèmes...)
4. Mise au point
   - Tests de non régression ( non associativité flottant)
   - Débogage (`gdb`, TotalView...)
   - Correction concurrence (Intel Thread Checker, Helgrind)
5. Optimisation
   - Profiling : Intel Thread Profiler, `gprof`, VTune...

Outline

Réduction (I)

Réduction $r = \bigoplus_{i=1}^{n} a_i$



	Séquentiel	Parallèle
Temps	$n-1$	$\lceil \log_2 n \rceil$
Opérateurs	1	$\lfloor \frac{n}{2} \rfloor$ ou $n-1$
Efficacité	1	$\frac{n-1}{\lfloor \frac{n}{2} \rfloor \lceil \log_2 n \rceil}$ ou $\frac{1}{\lceil \log_2 n \rceil}$

Parallélisme : gâcher utile !

Optimisations en binaire (I)

http://graphics.stanford.edu/~seander/bithacks.html
Quelques exemples trouvés dans virtualiseur QEMU qui doit implémenter des instructions de processeurs sur d'autre processeurs

- Trouver nombre de 0 binaires en tête d'un entier
  - Si le processeur sait le faire : utiliser instruction assembleur qui peut être cachée dans

```
1  T0 = __builtin_clz(T1);
```

  - Solution naïve en $\mathcal{O}(n)$ avec boucle

```
1  for (int i = sizeof(int)*CHAR_BIT - 1; i >= 0; i--)
     if ( (T1 & (1 << i)) != 0 )
3      break;
   T0 = sizeof(int)*CHAR_BIT - 1 - i;
```

  - Solution astucieuse $\mathcal{O}(\log n)$ (avec affectations conditionnelles...)

Optimisations en binaire (II)

```
1   /* Binary search for leading zeros.  */
2   T0 = 1;
    if ((T1 >> 16) == 0) {
4           T0 = T0 + 16;
            T1 = T1 << 16;
6   }
    if ((T1 >> 24) == 0) {
8           T0 = T0 + 8;
            T1 = T1 << 8;
10  }
    if ((T1 >> 28) == 0) {
12          T0 = T0 + 4;
            T1 = T1 << 4;
14  }
    if ((T1 >> 30) == 0) {
16          T0 = T0 + 2;
            T1 = T1 << 2;
18  }
    T0 = T0 - (T1 >> 31);
```

Optimisations en binaire *(III)*

- Exemple du comptage de population de 1 dans un entier SPARC64 `popc` nativement en $\mathcal{O}(\log n)$ réalisable par

```
1  uint32_t T0;

3  T0 = (T1 & 0x55555555) + ((T1 >>  1) & 0x55555555);
   T0 = (T0 & 0x33333333) + ((T0 >>  2) & 0x33333333);
5  T0 = (T0 & 0x0f0f0f0f) + ((T0 >>  4) & 0x0f0f0f0f);
   T0 = (T0 & 0x00ff00ff) + ((T0 >>  8) & 0x00ff00ff);
7  T0 = (T0 & 0x0000ffff) + ((T0 >> 16) & 0x0000ffff);
```

Environments with reductions

Available in various languages and libraries

- APL (1962 !): $+/$, $*/$, $\lceil/$, $\lfloor/$...
- Scilab & Matlab: `sum`, `prod`...
- MPI: `MPI_SUM`, `MPI_PROD`, `MPI_MIN`, `MPI_MAX`...
- Fortran: `SUM`, `PRODUCT`, `MINVAL`, `MAXVAL`...
- OpenMP

```
1       #pragma omp parallel for reduction(+:sum)
        for(i = 0; i <= 99; i += 1)
3          sum += i+k;
```

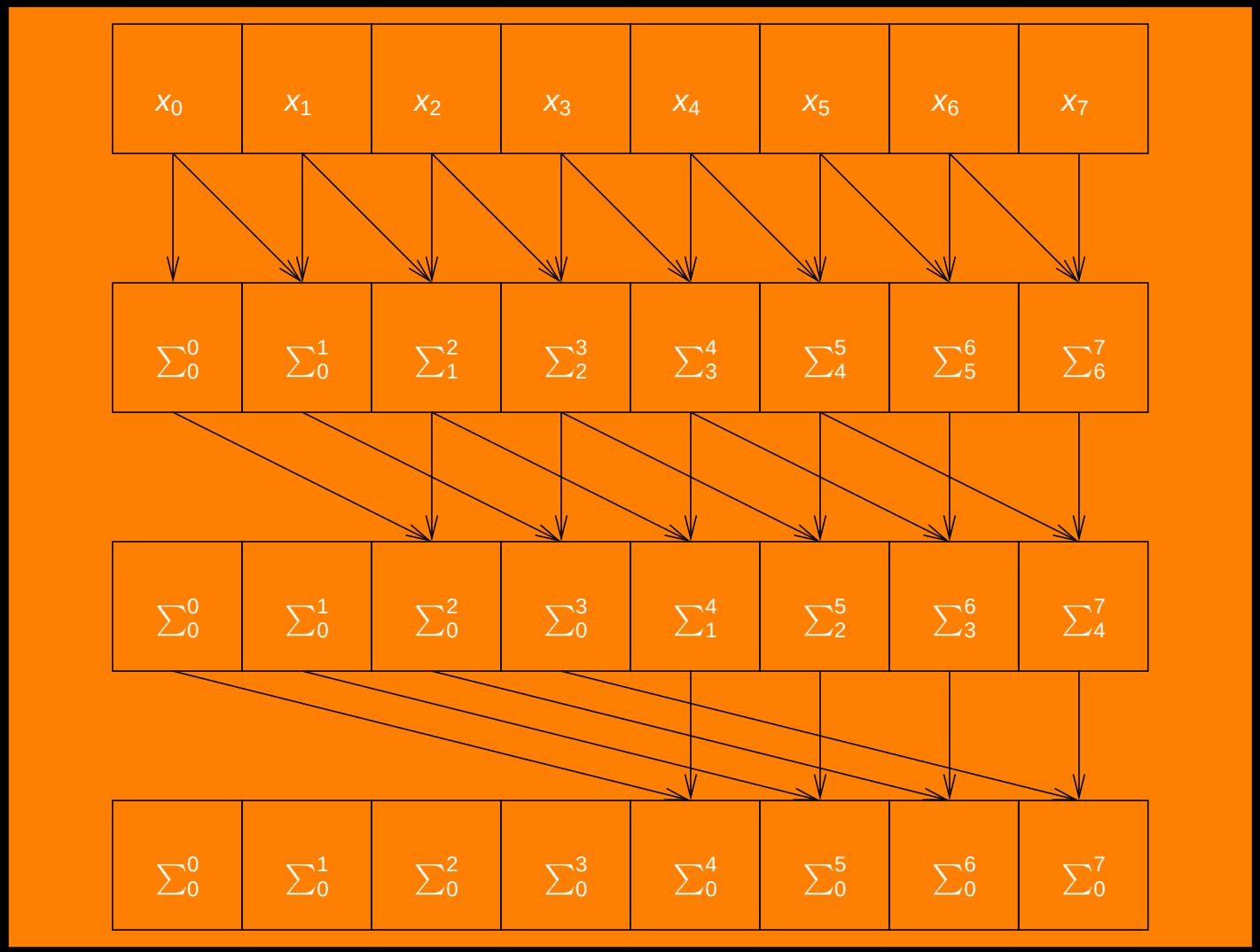
- C++ TBB `tbb::parallel_reduce()`
- Libraries for GPU: CuPP...

Opération préfixe parallèle (scan) *(I)*

Autre algorithmique parallèle classique (ILLIAC IV, 1968) :

$$\forall i \in [0, n-1], \quad S_i = \overset{i}{\underset{j=0}{\Big(+\Big)}} x_j$$

Opération préfixe parallèle (scan) *(II)*



Opération préfixe parallèle (scan) *(III)*

Si réutilisation des opérateurs :

	Séquentiel	Parallèle
Temps	$n-1$	$\lceil \log_2 n \rceil$
Opérateurs	1	$n-1$
Efficacité	1	$\frac{1}{\lceil \log_2 n \rceil}$

non associativité des opérations flottantes... Changement de l'ordre d'évaluation $\rightsquigarrow$ changement du résultat

Environments with parallel prefix/suffix scans

Available in various languages and libraries

- APL (1962 !): $+\backslash$, $*\backslash$, $\lceil\backslash$, $\lfloor\backslash$...
- Scilab & Matlab: `cumsum`, `cumprod`...
- MPI: `MPI_SCAN`, `MPI_PROD`, `MPI_MIN`, `MPI_MAX`
- Fortran: `SUM_PREFIX`/`SUM_SUFFIX`, `PRODUCT_PREFIX`/`PRODUCT_SUFFIX`, `MINVAL_PREFIX`/`MINVAL_SUFFIX`, `MAXVAL_PREFIX`/`MAXVAL_SUFFIX`...
- C++ TBB `tbb::parallel_scan()`
- Libraries for GPU: CuDPP `http://code.google.com/p/cudpp`

Suffix : begin at the end, reverse order

Parallel prefix variants

- Direction: prefix (left-to-right) or suffix (right-to-left)
- Can exclude the local element or not from the computation
  - $\text{scan}(+, (1,2,3,4)) = (1,3,6,10)$
  - $\text{scan}_{\text{excl}}(+, (1,2,3,4)) = (0,1,3,6)$
- Segmentation

x	1	2	3	4	5	6	7	8
s	?	t	f	t	t	f	f	t
$\text{scan}_{\text{seg}}(+, v, s)$	1	2	5	4	5	11	18	8

Compressing a vector (I)

- If lot of 0, useless to compute or store trivial values
- Store and compute useful values only: sparse representation & computations

Compress x into c according to validity v

x	42	?	7	8	?	?	3	?
v	1	0	1	1	0	0	1	0
$\text{scan}_{\text{excl}}(+, v)$	0	0	1	2	2	2	3	3

```
1 s = scan_add_exclude(x);
2 if (v[i])
    c[s[i]] = x[i];
```

c	42	7	8	3	?	?	?	?

Computing FIBONACCI suite (I)

$$u_{n+2} = u_{n+1} + u_n$$
$$u_1 = 1$$
$$u_0 = 0$$

- Naive recursion: $\mathcal{O}(2^n)$ 
- Naive recursion with memoization or loop: $\mathcal{O}(n)$ ☺

Computing FIBONACCI suite *(II)*

- Reduce the recursion distance

$$\begin{pmatrix} u_{n+2} \\ u_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} u_{n+1} \\ u_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

- Matrix exponentiation after diagonalization ⇝ gold number appears
  - Use KNUTH exponentiation algorithm $\mathcal{O}(\log n)$ ☺
  - If no symbolic computation: floating point approximation ⇝ not right integer result... ☹

Computing FIBONACCI suite *(III)*

- Inspect:

$$\begin{pmatrix} a+c & b+d \\ a & b \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Identify:

$$\begin{pmatrix} u_{n+2}+u_{n+1}=u_{n+3} & u_{n+1}+u_n=u_{n+2} \\ u_{n+2} & u_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} u_{n+2} & u_{n+1} \\ u_{n+1} & u_n \end{pmatrix}$$

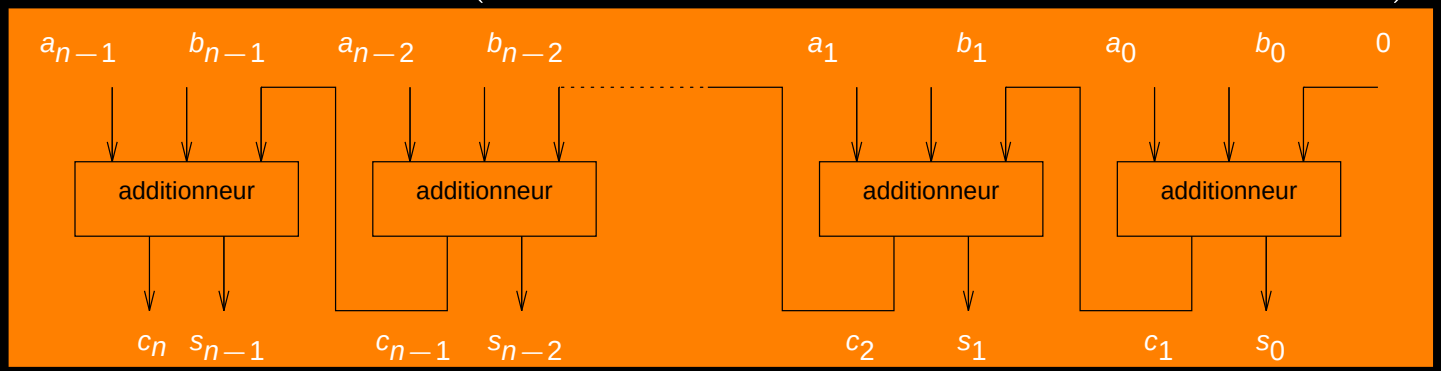
Recursion:

$$\begin{pmatrix} u_{n+1} & u_n \\ u_n & u_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

⇝ Apply matrix multiplication reduction and keep the upper left element: $\mathcal{O}(\log n)$ without diagonalization and floating point error computation! ☺

Parallélisme opérateur : addition entière *(I)*

Additionneur 1 bit : $(c, s) = \text{add}_1(a, b) = (a \wedge b, a \oplus b)$ (4 transistors)
Additionneur complet (3 entrées) :
$(c_o, s) = \text{add}(a, b, c_i) = \big((a \wedge b) \vee (a \wedge c_i) \vee (b \wedge c_i), a \oplus b \oplus c_i\big)$



Temps et complexité en $\mathcal{O}(n)$ pour additionneur n bits

Additionneur *carry-lookahead* *(I)*

Retenue = facteur limitant ⇝ changements de variable :

$$\left.\begin{array}{r} g_i = a_i b_i \\ p_i = a_i \vee b_i \end{array}\right\} \Longrightarrow \; c_{i+1} = g_i \vee p_i c_i$$

- si g_i est vrai alors $c_i + 1$ l'est : *génération* de la retenue
- si p_i est vrai alors si c_i est vrai elle est *propagée* à c_{i+1}.

$$\begin{aligned} c_{i+1} \quad = \quad & g_i \vee p_i g_{i-1} \vee p_i p_{i-1} g_{i-2} \vee p_i p_{i-1} p_{i-2} g_{i-3} \\ & \vee \cdots \vee p_i p_{i-1} \cdots p_1 g_0 \vee p_i p_{i-1} \cdots p_1 p_0 c_0 \end{aligned}$$

Appliquer une opération parallèle préfixe : temps en $\mathcal{O}(\log n)$ et espace en $\mathcal{O}(n \log n)$.
Autres méthodes : *carry skip*, *carry select*,...
Soustraction : inverser une des entrée ($\overline{a}$ ou $\overline{b}$) et $c_0 = 1$

Multiplication entière (I)

Méthode moyenâgeuse avec table de carrés (mémoire morte, place en $\mathcal{O}(n)$ au lieu de $\mathcal{O}(n^2)$) :

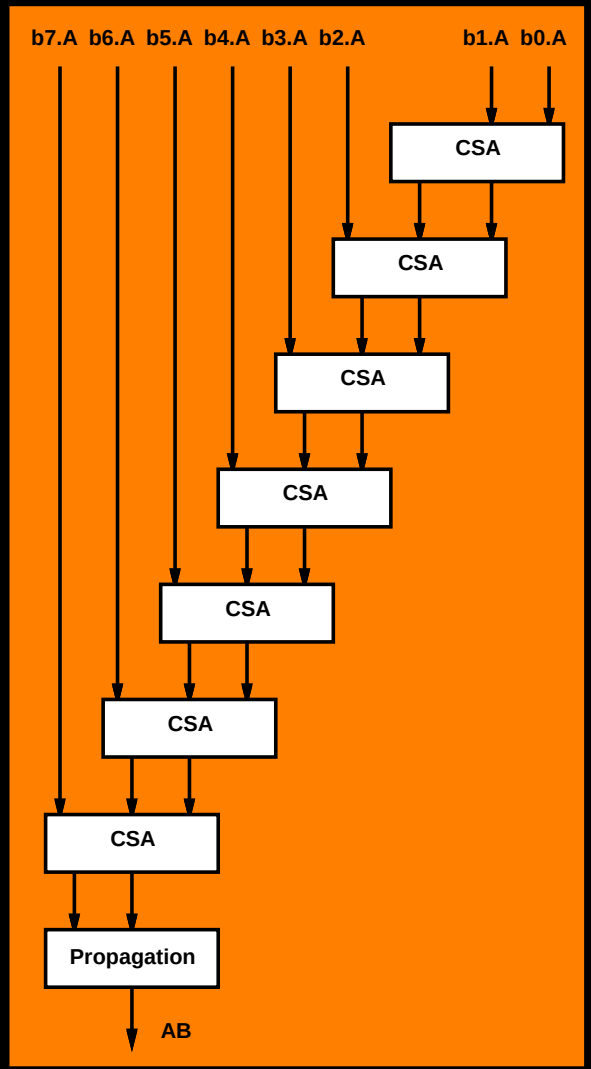
$$a \times b = \frac{(a+b)^2 - a^2 - b^2}{2}$$

Bonne vieille méthode manuelle : n additions :

- Sauter les bits à 0 (temps de multiplication non constant)
- Propagation de la retenue lente $\rightsquigarrow$ ne pas propager et garder toutes les retenues (*Carry Save Adder*) et propager lors de la dernière addition
  Utilisation d'un codage redondant ($2 \times n$ bits, chiffres dans 0–3) sauf pour le dernier résultat :

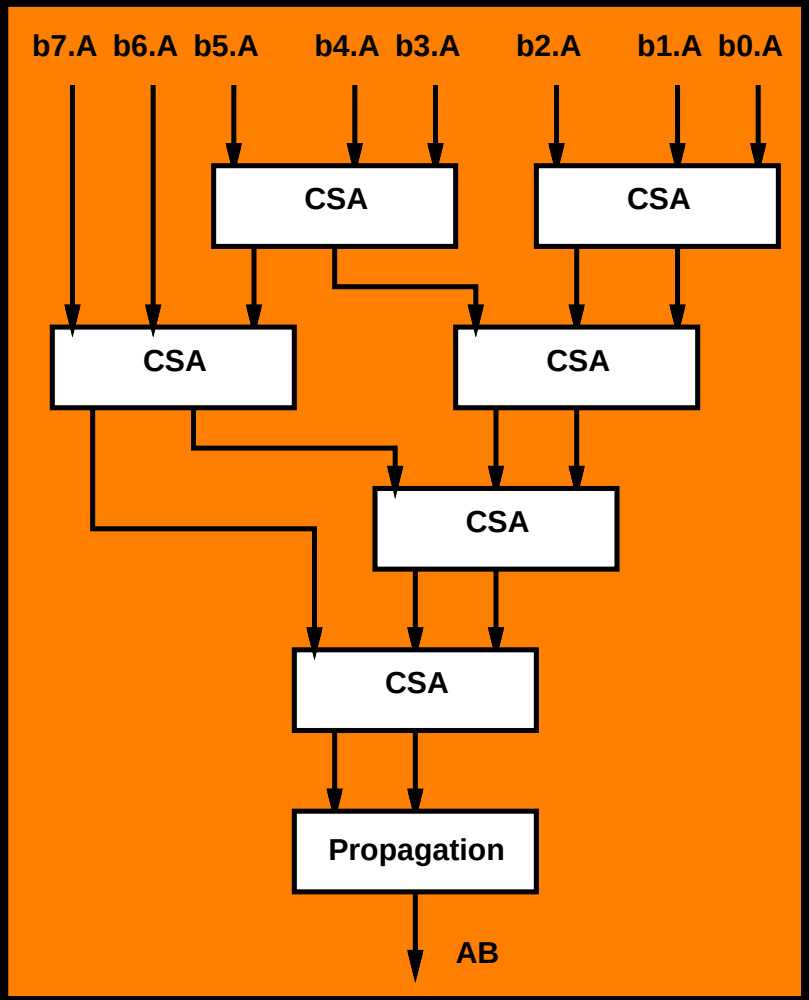
$$(s_i, c'_i) = (\lfloor (a_i + b_i + c_i)/2 \rfloor, \lfloor (a_i + b_i + c_i) \mod 2 \rfloor)$$

Multiplication entière *(II)*



Multiplication par arbre de Wallace

Réduction ⇝ arbre :



Retour à l'addition entière

Parsing a regular language (I)

« Update to "data parallel algorithms" », W. Daniel Hillis & Guy L. Steele, Jr. Communications of the ACM, Volume 30 Issue 1, Jan. 1987

How to parse `if x<=n then print("x = ", x);` into

`if` | `x` | `<=` | `n` | `then` | `print` | `(` | `"x = "` | `,` | `x` | `)` | `;` ?

Use finite-state automaton with transition from current state to new one according to character class

- N: initial state
- A: start of alphabetic token
- Z: continuation of alphabetic token
- *: single-special-character token
- <: < or > character
- =: = following < or >
- Q: double quote starting a string

Parsing a regular language *(II)*

- S: character within a string
- E: double quote ending a string

Old state	Character read										
	A	...	Z	+	-	*	<	>	=	"	space/new line
N	A	...	A	*	*	*	<	<	*	Q	N
A	Z	...	Z	*	*	*	<	<	*	Q	N
Z	Z	...	Z	*	*	*	<	<	*	Q	N
*	A	...	A	*	*	*	<	<	*	Q	N
<	A	...	A	*	*	*	<	<	=	Q	N
=	A	...	A	*	*	*	<	<	*	Q	N
Q	S	...	S	S	S	S	S	S	S	E	S
S	S	...	S	S	S	S	S	S	S	E	S
E	A	...	A	*	*	*	<	<	*	S	N

Parsing a regular language *(III)*

- Consider characters as function mapping an automaton state into an other one ☺: NY is character Y applied to state N and produce state A
- N`x<=y` = A`<=y` = `<=y` = `=y` = A
- The composition operation is associative...
- The character function can be represented by an array indexed by state with produced state as value
- Perform parallel prefix with combining functions of the string ☺
- Use initial automaton state to index into *all* these arrays: every character has been replaced by the state the automaton would have after that character

Can be generalized to any function that can be reasonably represented with a look-up-table

CUDA CuDPP

Libraries for GPU: CuDPP `http://code.google.com/p/cudpp`

- RadixSort
- Array compaction
- Scan with C++ template operator
- Segmented scan
- Sparse matrix-vector multiply

Use plans *à la* FFTW

Cloud & MapReduce

- MapReduce
  - Map: $y(i) = f(y(i))$
  - Reduce: $z = \bigoplus_i y(i)$
    Sorting is also a parallel prefix operation...
- Various implementations for cloud computing: Apache Hadoop (Java) with some file system support (HDFS)...
- Good throughput and scalability but bad latency 

Outline

Applications codage binaire : multispin-coding *(I)*

- Exploitation du parallélisme en bits
- Ranger plusieurs petites données par mot machine
- 4 opérations sur 64 bits/cycle ≡ 256 opérations sur 1 bit/cycle !
- Opérations binaire style `^`, `&`, `|`, `~` sans problème
- Jeux d'instructions :
  - 1 Alpha 21164 à 600 MHz ≡ 76,8 GIPS 1 bit, 9,6 GIPS 8 bits
  - 1 Pentium 4 SSE3 à 4 GHz : 2 opérations 128 bits/cycle ≡ 1 TIPS (10^{12} opérations par secondes) 1 bit/cœur
- Idée : plutôt que de résoudre 1 problème à la fois, éclate problème en binaire pour calculer 256 tranches de problèmes binaires à la fois en AVX

Exemple : bibliothèques de cassage de codes cryptographiques (détection mots de passe faibles avec John the Ripper), traitement d'image, traitement du signal, codage, optimisation de programmes...

Application utilisant des additions 9 et 6 bits *(I)*

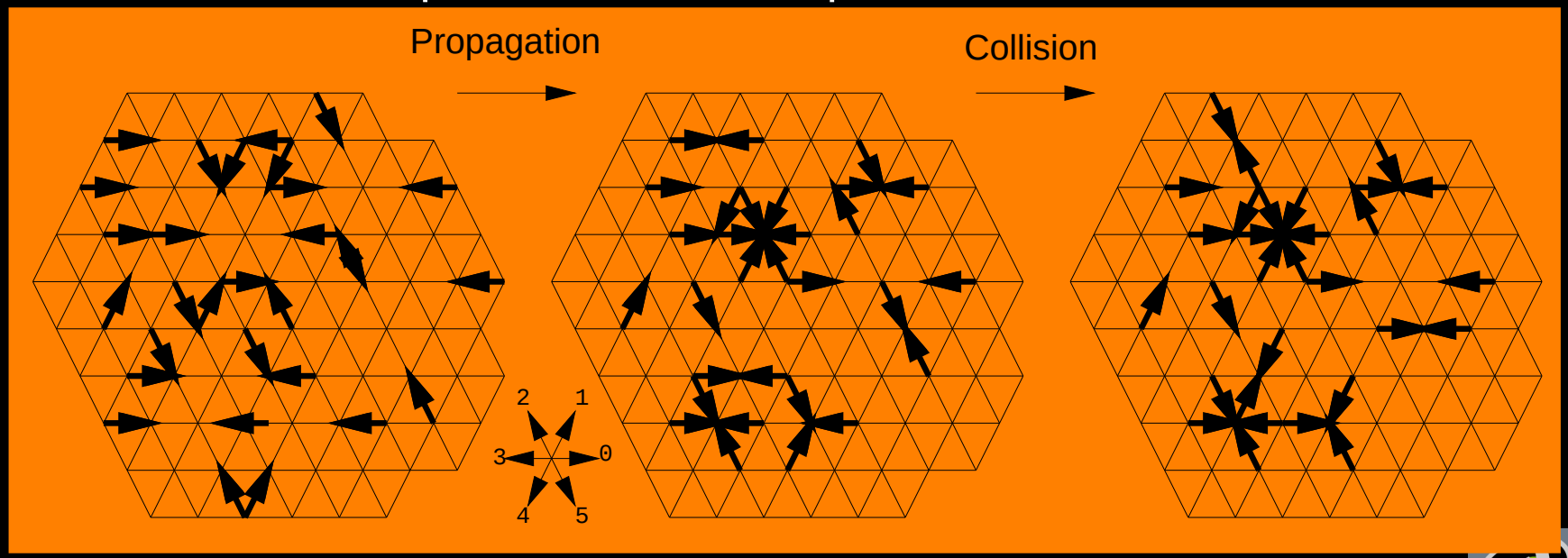
- Compactage dans 32 bits `a_xxs_yys` :

`quality`	`q_a`	0	`q_x`	0	`q_y`
8	6	1	8	1	8

- `q_a` sur 6 bits, `q_x` et `q_y` sur 8 bits
- Opérations sur `q_x` et `q_y` sur 9 bits $\rightsquigarrow$ stockage sur 9 bits aussi (évite l'extraction)
- Garde des 0 délimiteurs absorbant les retenues (`& masque`)
- Besoin de tester $(q_x, q_y) \in [-128, 127]^2$:
  - Changement repère (biais $+128$) $\rightsquigarrow$ $(q'_x, q'_y) \in [0, 255]^2$
  - Test de `a_xxs_yys & ((1<<8) + (1<<18)) == 0` : 1 instruction !

Gaz sur réseau (I)

- Méthode *lattice* BOLTZMAN
- Sites contenant des particules se déplaçant quantiquement
- Interactions entre particules sur chaque site



- Tableau de sites contenant 1 bit de présence d'1 particule allant dans 1 direction
- Symétrie triangulaire

Gaz sur réseau (II)

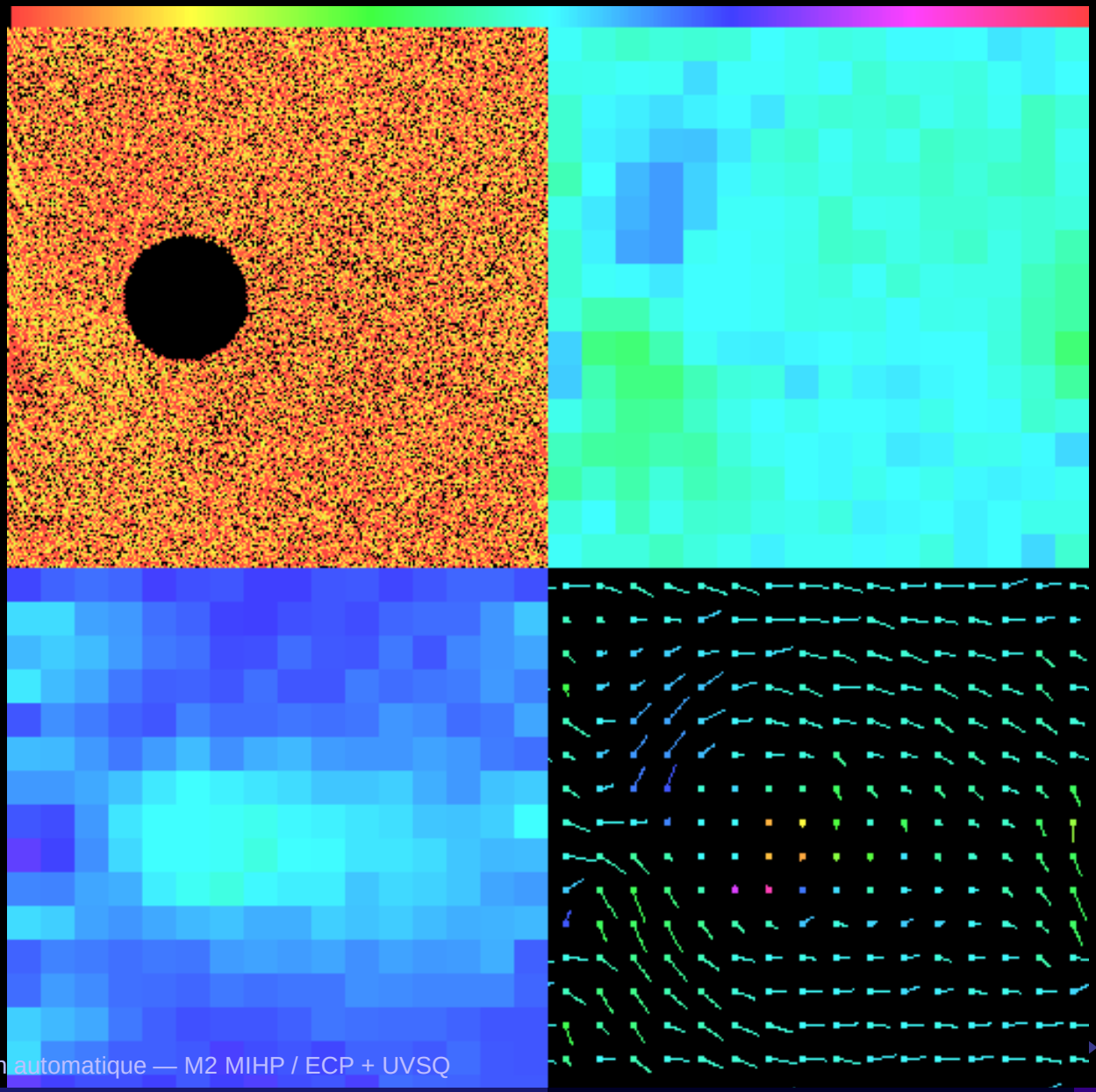
- Compactage de 32 ou 64 sites/`int` par direction

```
a = lattice[RIGHT];
b = lattice[TOP_RIGHT];
c = lattice[TOP_LEFT]; // Particules qui montent à gauche
d = lattice[LEFT]; // Particules qui vont à gauche
e = lattice[BOTTOM_LEFT];
f = lattice[BOTTOM_RIGHT];
s = solid; // Une condition limite
ns = ~s;
r = lattice[RANDOM]; // Un peu d'aléa
nr = ~r;
    /* A triplet ? */
triple = (a^b)&(b^c)&(c^d)&(d^e)&(e^f);
    /* Doubles ? */
double_ad = (a&d&~(b | c | e | f));
double_be = (b&e&~(a | c | d | f));
double_cf = (c&f&~(a | b | d | e));
    /* The exchange of particles : */
change_ad = triple | double_ad | (r&double_be) | (nr&double_cf);
```

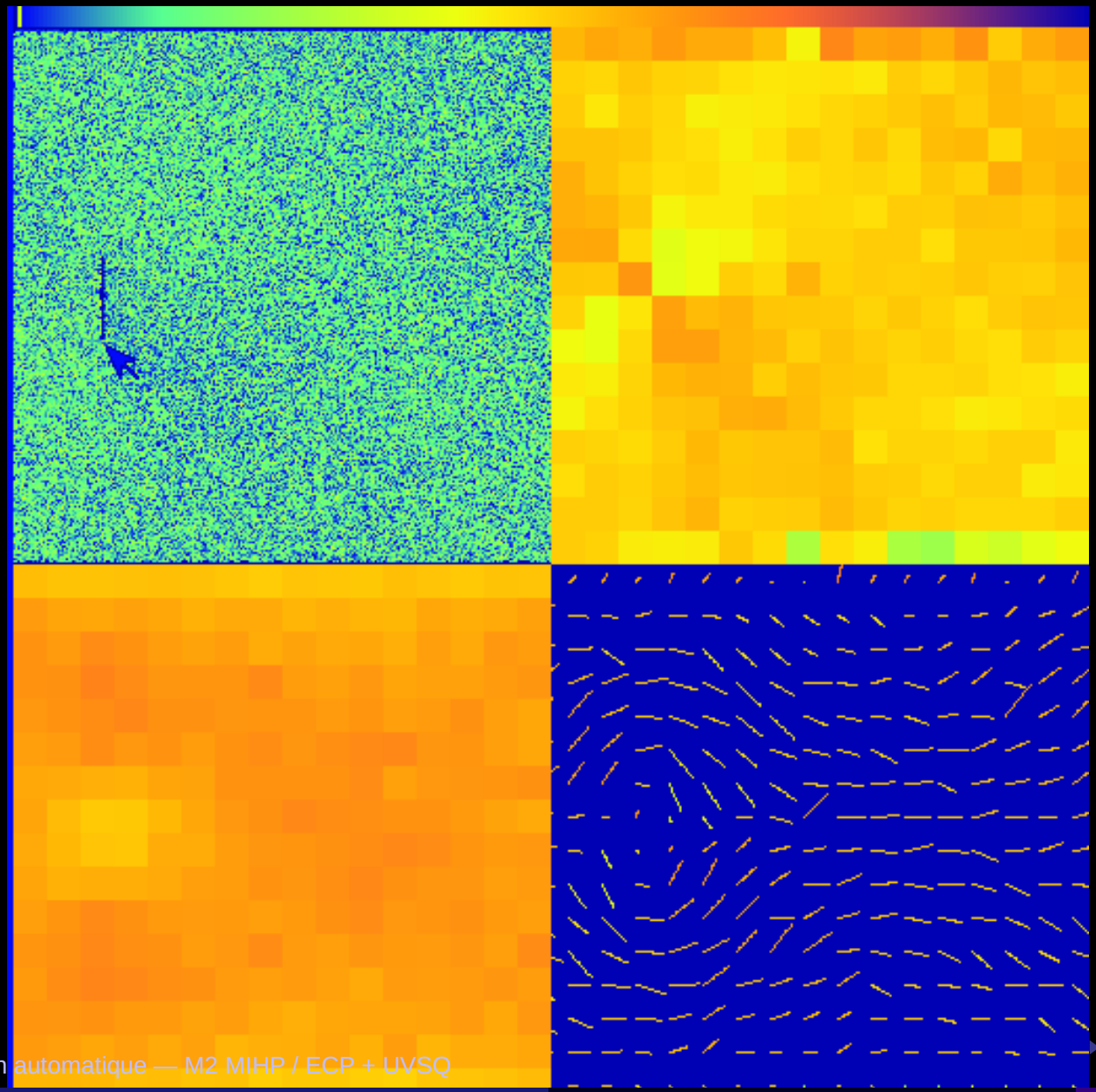
Gaz sur réseau (III)

```
change_be = triple | double_be | (r&double_cf) | (nr&double_ad);
change_cf = triple | double_cf | (r&double_ad) | (nr&double_be);
    /* Where there is blowing, collisions are no longer valuable : */
bl = blow[N_DIR];
s &= ~bl;
ns &= ~bl;
    /* Effects the exchange where it has to do according the solid : *
lattice[RIGHT] = (((a^change_ad)&ns) | (d&s))
    | bl&blow[RIGHT];
lattice[TOP_RIGHT] = (((b^change_be)&ns) | (e&s))
    | bl&blow[TOP_RIGHT];
lattice[TOP_LEFT] = (((c^change_cf)&ns) | (f&s))
    |bl&blow[TOP_LEFT];
lattice[LEFT] = (((d^change_ad)&ns) | (a&s))
    | bl&blow[LEFT];
lattice[BOTTOM_LEFT] = (((e^change_be)&ns) | (b&s))
    | bl&blow[BOTTOM_LEFT];
lattice[BOTTOM_RIGHT] = (((f^change_cf)&ns) | (c&s))
    | bl&blow[BOTTOM_RIGHT];
```

Gaz sur réseau — Cylindre



Gaz sur réseau — Instabilités de Von KARMAN



Conclusion sur multispin coding

- Méthodes de simulation permet de modéliser de nombreux phénomènes (agrégations, gaz + fluides, solides, biologie...)
- Marche avec n'importe quelle architecture : parallélisme de bits des opérations binaires
- Passage en AVX ou LRBni : 256 ou 512 sites traités par cycles
- Bien pour cartes graphiques aussi
- Autre méthode par table de collision mais problème débit mémoire/cache (cf scatter/gather des processeurs vectoriels)
  - Bien pour cartes graphiques si utilisation de caches possible

Outline

Nombres flottants (I)

- Besoin de représenter des valeurs plus « continues » et plus de dynamique que types entiers
- Sous-ensemble de $\mathbb{D}(\subset \mathbb{R})$
- Représentation souvent au format IEEE 754-1985

$$f = (-1)^{\mathcal{S}} \times \mathcal{M} \times 2^{\mathcal{E}}$$

  - $\mathcal{S}$: bit de signe
  - $\mathcal{M}$: mantisse (entier positif)
  - $\mathcal{E}$: exposant
- Plusieurs tailles de flottants
  - Simple précision (**float**)
  - Double précision (**double**)
  - Précision étendue (**long double**)

Nombres flottants (II)

- Codage en mémoire

	Taille en bit			
Format	Totale	Signe	Exposant	Mantisse
Simple	32	1	8	24
Double	64	1	11	53
Étendu	≥ 80	1	≥ 15	≥ 64

  - $1 + 8 + 24 = 33 \neq 32$: voir plus tard...
  - Exposant codé en biaisé : $\mathcal{E}_{\text{réel}} = \mathcal{E}_{\text{stocké}} - \mathcal{E}_{\text{biais}}$
  - Tri lexicographique sur bits compatible avec tri flottant ! Même si on ne gère pas le flottant on sait trier ☺

Format	Minimum en dénormalisé	Minimum en normalisé	Maximum fini	2^{-N} (grain)	Chiffres significatifs
Simple	$1,4 \cdot 10^{-45}$	$1,2 \cdot 10^{-38}$	$3,4 \cdot 10^{38}$	$5,96 \cdot 10^{-8}$	6–9
Double	$4,9 \cdot 10^{-324}$	$2,2 \cdot 10^{-308}$	$1,8 \cdot 10^{308}$	$1,11 \cdot 10^{-16}$	15–17
Étendu	$\leq 3,6 \cdot 10^{-4951}$	$\leq 3,4 \cdot 10^{-4932}$	$\geq 1,2 \cdot 10^{4932}$	$\leq 5,42 \cdot 10^{-20}$	$\geq$ 18–21

Nombreux paramètres définis dans `<float.h>`

Nombres flottants (III)

- Possibilité de déclencher exceptions (division par 0, débordement,...) (fonction exécutée sur événement)
- Rajout de quantités symboliques (déclarées dans `<math.h>`)
  - $+0$ et -0. Néanmoins $+0 = -0$ est vrai
  - $+\infty$ et $-\infty$ (par exemple $\frac{1}{+0}$ et $\frac{1}{-0}$) (`HUGE_VAL`...)
  - NaN (*Not a Number*) pour $\frac{0}{0}$ ou $\sqrt{-1}$
    Seul cas ou $x \neq x$ lorsque x vaut NaN. Existe en signé et en version déclenchant exception (SNaN)

  ⇝ Peuvent simplifier programmation et calcul si bien géré (éviter tests cas particuliers...)
- ∃ Nombreux choix d'arrondi (plus proche, +, -, vers 0,...)
- `http://grouper.ieee.org/groups/754` En cours de révision `http://en.wikipedia.org/wiki/IEEE_754r` si vous voulez participer ☺

Conversion flottants→entiers à l'arrache (I)

- En temps normal `int i = (int) f` doit faire le travail
- ∃ nombreuses fonctions de conversion et d'arrondi dans la bibliothèque mathématique (`rint()`, `round()`...)

La bibliothèque du C permet de le faire tout seul mais pas toujours disponible (cf. microcontrôleur Coupe de Robotique 2008). Pour hackers :

```
1  int ftoi(float f)
2  {
       // Récupère les bits du flottant dans un entier :
4      uint32_t dw = *((uint32_t *) &f);
       // La valeur spéciale où tous les bits sont à 0 code 0 :
6      if (dw == 0)
           return 0;
8      // Récupère l'exposant codé sur 8 bits et compense le biais :
       char exp = (dw >> 23) - 127; // Suppose un char de 8 bits
10     if (exp < 0 || exp > 23)
           /* Si l'exposant est négatif, de toute manière, le nombre vaut
12            moins que 1, donc arrondi à 0. Si c'est supérieur à 23, le nombr
              est supérieur à 2^{24} en on décide de le jeter et de répon
```

Conversion flottants→entiers à l'arrache *(II)*

```
14              arbitrairement 0. Mmmm... On pourrait gérer jusqu'à 2^{32} mais
                faudrait corriger le code ci-après */
16          return 0;
        /* Construit le nombre avec le 1 de poids fort qui est économisé dans
18          la norme IEEE-754, puis les 23 autres bits de la mantisse cadrés
            en fonction de l'exposant : */
20      int val = (1 << exp) + ((dw & 0x7FFFFF) >> (23 - exp));
        // En fonction du bit de signe, inverse le résultat :
22      if (dw & 0x80000000)
            return -val;
24      else
            return val;
26  }
```

Bon, évidemment, ceci ne gère pas toute la norme, le dénormalisé, les infinis, etc.

Nombres flottants $\neq$ réels ! *(I)*

« What Every Computer Scientist Should Know About Floating-Point Arithmetic », David GOLDBERG, Computing Surveys, mars 1991, ACM

- Nombres flottants $\equiv$ pale imitation de $\mathbb{R}$ et même de $\mathbb{D}$ ☹
- Nombreuses approximations
-  Propriétés algébriques de $\mathbb{R}$ non vérifiées : non associatif

$$(1 \oplus 10^{40}) \ominus 10^{40} = 0$$
$$1 \oplus (10^{40} \ominus 10^{40}) = 1$$

- Changement des résultats possibles selon optimisations... ☹
- Notion d'équivalence séquentielle de programme entre différentes versions

Nombres flottants $\neq$ réels ! *(II)*

  - Forte : le programme obtenu donne le même résultat
  - Faible : le programme obtenu donne le même résultat modulo les problèmes numériques précédents
- Choisir programmation prenant en compte ces caractéristiques
  - TEX écrit en virgule fixe 16+16 bits pour portabilité multi-plateforme ☺
  - Compromis entre performances & précision
-  Compilateurs devraient en tenir compte (pas optimisations sauvages)
- Exemples
  - $(x - y)(x + y)$ plus précis (voire plus rapide) que $x^2 - y^2$
  - Algorithme somme de flottants

Algorithme de sommation de flottants (I)

- Solution triviale

```
1  double x[N];
2  double s = 0;
   for(int i = 0; i < N; i++)
4    s += x[i];
```

$$s = \sum_{i=0}^{N-1} x_i(1+\delta_i)$$

avec $|\delta_i| < (N-i)\epsilon$

Algorithme de sommation de flottants *(II)*

- Version KAHAN

```
1  double x[N];
2  double s = x[0];
   double c = 0;              // Erreur d'arrondi
4  for(int i = 1; i < N; i++) {
     double y = x[i] - c;     // Compense erreur précédente
6    double t = s + y;        // Nouvelle somme
     c = (t - s) - y;         // Estime l'erreur arrondi
8    s = t;
   }
```

$$s = \sum_{i=0}^{N-1} x_i(1+\delta_i) + \mathcal{O}(N\epsilon^2 \sum_{i=0}^{N-1} |x_i|) \quad \text{avec} \quad |\delta_i| \leq 2\epsilon$$

Optimisations incontrôlées du programme fait des ravages ici... car revient à algorithme trivial ! ☹

Vers des nombres flottants normalisés (I)

- Possible de représenter des nombres de plusieurs manières

$$\mathcal{M}' = 2^{-a}\mathcal{M}$$
$$\mathcal{E}' = \mathcal{E} + a$$

- Problème des codages redondants : comparaisons difficiles ☹
- Idée 1 : normaliser ! Exemple : choisir le $\mathcal{M}$ le plus grand pouvant loger dans les bits alloués pour la mantisse
- Idée 2
  - $\forall \mathcal{M} \neq 0$: commence toujours par 1 en binaire
  - ⇝ Ne pas stocker ce 1 évident...

⇝ Flottant normalisé : gagne 1 bit de précision pour la mantisse ! ☺

Pourquoi des nombres flottants dénormalisés ? *(I)*

- Soustraction de 2 nombres normalisés, par exemple en simple précision

$$a = 2,05 \cdot 10^{-37}$$
$$b = 2,03 \cdot 10^{-37}$$
$$a - b = 2 \cdot 10^{-39}$$
$$a \ominus b = 0$$
$$a \neq b$$

  Seule solution car $\mathcal{M}$ ne peut pas commencer par 1... ☹
- Idée : rajouter mode dénormalisé pour très petits nombres où $\mathcal{M}$ peut ne pas commencer par un 1
- Permet *underflow* (dépassement de capacité par le bas) progressif

Pourquoi des nombres flottants dénormalisés ? *(II)*

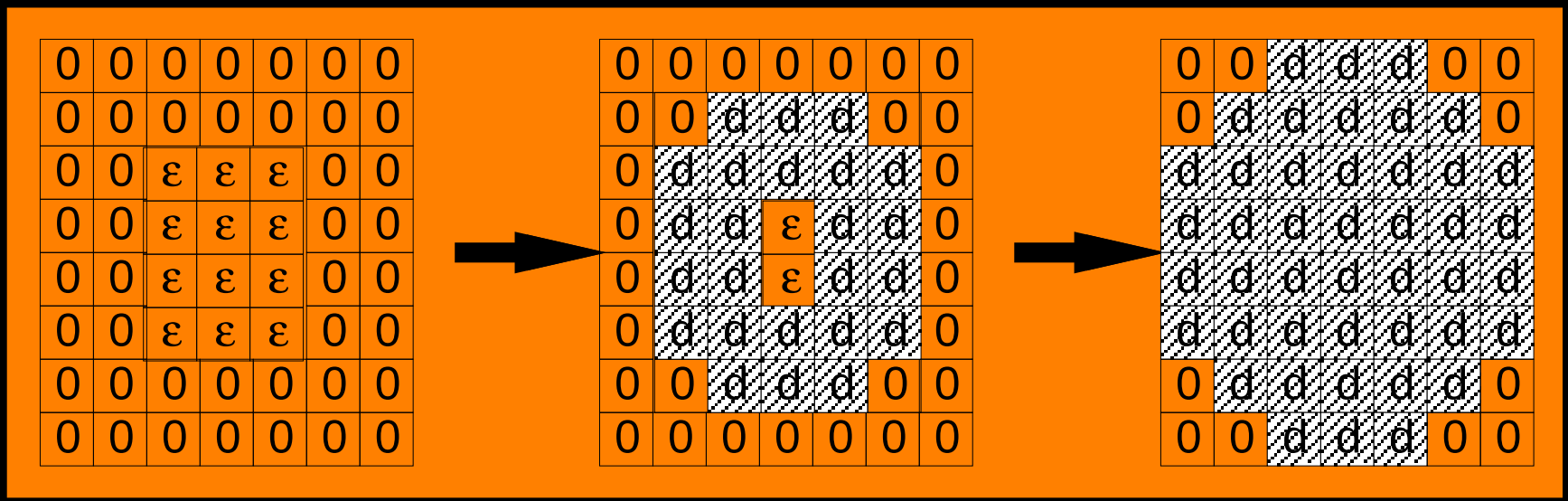
-  Si flottant dénormalisé non géré directement en matériel : génère exception et calculs terminés par... système d'exploitation $\rightsquigarrow$ performances $\searrow\searrow$ ☹
-  Parfois autorisation exception $\Longrightarrow$ suppression pipeline (DEC Alpha) ☹

Dénormalisation flottante (I)

- Parfois exception IEEE-754 générée lors de la dénormalisation
- Typiquement un programme de différences finie avec un domaine avec de petites valeur ϵ entouré de 0:

$$x^n_{i,j} = \frac{x^v_{i-1,j} + x^v_{i+1,j} + x^v_{i,j-1} + x^v_{i,j+1}}{4}$$

⇝ Propagation d'ondes de dénormalisation



Dénormalisation flottante *(II)*

- À pleurer sur machine parallèle si ordonnancement statique : tous les processeurs attendent le plus lent ! ☹
- Rajout d'un biais pour ne plus être au voisinage de 0. Mais perte de dynamique... Compromis

$$y^n_{i,j} = x^n_{i,j} + b \quad (1)$$

$$y^n_{i,j} = \frac{y^v_{i-1,j} + y^v_{i+1,j} + y^v_{i,j-1} + y^v_{i,j+1}}{4} \quad (2)$$

- Bonne nouvelle : GPU gèrent les nombres dénormalisés en matériel sans pénalité

Outline

Synchronisation (I)

- Exemple de 2 processus producteurs qui produisent 1 produit à chaque fois et 1 variable globale pour compter ce qu'on produit

P_1 :

$$\texttt{produits} = \texttt{produits} + 1$$

P_2 :

$$\texttt{produits} = \texttt{produits} + 1$$

Supposons que `produits` = 42 et que P_1 et P_2 s'exécutent en même temps, on peut avoir P_1 et P_2 qui lisent 42 et réécrivent... 43 au lieu d'avoir 44 ☹

↝ Besoin d'*atomicité* ou d'avoir de l'exclusion mutuelle: protéger `produits` contre des accès chaotiques

- Besoin de coordonner différents processus pour coopérer sans erreur

Synchronisation (II)

- Sinon, on a des situations de compétitions (*race conditions*)

∃ nombreux moyens techniques d'assurer ce genre de protection

Sémantique mémoire parfois étrange...

Shared memory semantics (I)

- Sequential classical assumption in a multiprocessor: memory access are... in order (causality)
- May not be true for multiprocessor!
- Read and write access done through queues for efficiency, that may reverse some access
- The sequential approximation is the basis of some synchronization algorithms
- ⇝ Need for a precise semantics about global memory behaviour...
- ... But processor behaviour not always well defined
- « *Memory Models: A Case for Rethinking Parallel Languages and Hardware* », Sarita V. Adve, Hans-J. Boehm. *Communications of the ACM* Vol. 53 No. 8, August 2010, pages 90-101

Shared memory semantics *(II)*

- « *x86-TSO: A Rigorous and Usable Programmer's Model for x86 Multiprocessors* », Peter Sewell, Susmit Sarkar, Scott Owens, Francesco Zappa Nardelli, Magnus O. Myreen. *Communications of the ACM* Vol. 53 No. 7, July 2010, pages 89-97

DEKKER's mutual exclusion algorithm (I)

- First known correct solution to mutual exclusion in ≈ 1960

```
1  // Scoreboard to mark work interest per process. Initial state:
   bool volatile flag[2] = {false, false};
3  // The priority if any conflict:
   int volatile turn = 0;
5  // From process p = 0 or 1
   void acquire(int p) {
7       flag[p] = true; // Warn that I want to work
        while (flag[1 - p]) { // While the other process wants to work
9           if (turn != p) { // If it is not my turn
                flag[p] = false; // Give up and be polite...
11              while (turn != p) { // Wait the other one to finish
                }
13              flag[p] = true; // Warn again I want to work and retry
            }
15      }

17 // Some critical section in between...

19 void leave(int p)
        turn = 1 - p; // Be polite: give the other one a try on next conflict
21      flag[p] = false; // My work is finished
   }
```

DEKKER's mutual exclusion algorithm *(II)*

- Do not need any special instruction to work (test-and-set, atomic memory swap...)
- But need to warn the compiler that some optimizations such as constant propagation is forbidden in this case to avoid this optimization:

```
1  ...
2  int register constant = flag[1 - p]; // Loop invariant hoisting
   while (constant) { // Infinite loop!
4  ...
   }
6  ...
```

↝ Need volatile keyword to warn about external world 

Sequential consistency... from theory to practice *(I)*

- Often considered implicitly true by programmers
- In DEKKER's algorithm, with initial state x =flag[0] = 0 and y =flag[1] = 0

Processor 0	Processor 1
mov (x)←1 mov eax←(y)	mov (y)←1 mov ebx←(x)

should not produce $eax_{p0} = 0 \wedge ebx_{p1} = 0$ in... theory (2 process in the critical section)

- Theoretical sequential-consistent interleaving of instructions (LAMPORT 1979), eventually considered as atomic

(x)←1 eax←(y) (y)←1 ebx←(x)	(x)←1 (y)←1 eax←(y) ebx←(x)	(x)←1 (y)←1 ebx←(x) eax←(y)	(y)←1 (x)←1 eax←(y) ebx←(x)	(y)←1 (x)←1 ebx←(x) eax←(y)	(y)←1 ebx←(x) (x)←1 eax←(y)
eax=0 ebx=1	eax=1 ebx=1	eax=1 ebx=1	eax=1 ebx=1	eax=1 ebx=1	eax=1 ebx=0

Sequential consistency... from theory to practice (II)

We cannot have $eax_{p0} = 0 \wedge ebx_{p1} = 0$ in... theory

- But in practice
  - Read are write are done through processor-local queues for memory latency pipelining
  - Since x and y have different addresses, there is no *local* causality violation
  - But write commit to global memory or cache may be delayed... breaking global causality ☹

  ⇝ In practice we may have $eax_{p0} = 0 \wedge ebx_{p1} = 0$ ⇝ 2 process in critical section  ☹
- Hardware architects may not envision all programming consequences of some hardware optimizations...
- Modern shared multiprocessors have no sequential memory semantics 
- ∃ Benchmarks to detect some of this ill-behaviours (Litmus...)

Data-race-free memory model (I)

- Many different memory model exists: ADA, OpenMP, Java, C++,
- Need a model understandable at least by... expert programmers! ☹
- At least, for programs without data race, sequential model should stand: *data-race-free memory model*
  - A data race occurs when 2 threads share data with at least one write
  - Only care about parts with data race
  - Parts without data race should behave with a sequential memory semantics
-  Changing a memory model on an architecture means
  - Changing programming 
  - Breaking memory compatibility  
- Well defined semantic for SPARC processors (old parallel machines ☺)

Data-race-free memory model *(II)*

- More precise models for some *fence* instructions enforcing some memory ordering have been added to recent AMD (2007, 2009) and Intel (2007, 2008, 2009) specifications

Processor 0	Processor 1
mov (x)←1	mov (y)←1
fence	fence
mov eax←(y)	mov ebx←(x)

- ∃ old `LOCK` prefix instruction on *x*86 that locks a global memory lock and flush the local write buffer, prevent other write buffer flush, but quite slow (memory latency and global big lock)...

Processor 0	Processor 1
lock mov (x)←1	lock mov (y)←1
mov eax←(y)	mov ebx←(x)

Data-race-free memory model (III)

- ⇝ Synchronization stuff should be implemented by specialists in language constructs and libraries for the programming masses...
  - Java `volatile`
  - `atomic` in next C++ and C version
  - OpenMP `flush` pragma
- Safe language issues
  - Safe languages allow some safe behaviour by construction: Java with sand-boxed execution for untrusted code
  - What if a multithread untrusted code has a wicked race condition *by design*?
  - Is it practically possible to build security breach with race conditions?

◀ Go back to cache coherence protocols

Outline

Outline

Message Passing Interface (MPI) *(I)*

- Le passage de message pour les nuls ! ☺
- Bibliothèque de fonctions de communication disponible pour de nombreux langages et systèmes d'exploitation
- Portabilité et nivellement par le bas : programmation de SMP aussi en MPI...
- Ressources
  - `http://en.wikipedia.org/wiki/Message_Passing_Interface`
  - MPI Forum `http://www.mpi-forum.org`
  - MPICH : A Portable Implementation of MPI `http://www-unix.mcs.anl.gov/mpi/mpich/`
  - LAM / MPI Parallel Computing `http://www.mpi.nd.edu/lam/`
  - MPE Graphics–Scalable X11 Graphics in MPI `http://www-fp.mcs.anl.gov/~lusk/papers/mpe/`
  - Livre « Using MPI: Portable Parallel Programming with the Message-Passing Interface » `http://www-unix.mcs.anl.gov/mpi/usingmpi/`

Message Passing Interface (MPI) *(II)*

- Faute de temps je n'utilise plus mes transparents trop complets
  `http://enstb.org/~keryell/cours/MR2/IAHP/MPI`
- J'utilise « An Introduction to MPI Parallel Programming with the Message Passing Interface » de William Gropp & Ewing Lusk
  `http://www-unix.mcs.anl.gov/mpi/tutorial/mpiintro/MPIIntro.PPT`

Bibliothèques systèmes (I)

- Proches du système d'exploitation
- Contrôle fin de tous les paramètres : priorité, affinité tâche/processeur...
- Posix normalise une API système proche d'Unix. Contient aussi des threads
  - Windows a aussi une API Posix
  - ∃ version libre des *pthreads* pour Windows
    `http://sourceware.org/pthreads-win32`
  - Assure portabilité multi-OS
- `libnuma` sous Linux pour contrôler comportement mémoire sur machine à *Non Uniform Memory Access* : allocation sur bancs, nœuds, partie de la machine, locale à 1 thread...

libnuma

- Non-Uniform Memory Access (NUMA): multiprocessor systems with memory divided into multiple memory nodes
  - Access time depends on the relative location
  - Need a way to select where the memory is allocated
    - Local
    - On a given node
    - Interleaved on a node subset
    - By preference on some nodes
- Low level Linux support (began with SGI efforts)
  - `/proc/.../maps` describes the normal memory map
  - `/proc/.../numa_maps` added NUMA information
  - `sys/devices/system/node` describes nodes
  - `get_mempolicy()` get the default policy of the process or an address range
  - `set_mempolicy()` set the default policy
  - `mbind()` set the policy for a given address range
  - `move_pages()` move some pages on some nodes

libnuma (II)

- `migrate_pages()` move all pages from a process on some nodes to some other nodes (node maintenance...)
- `getcpu()` give CPU and node id of the calling thread ( preemption...)

- `libnuma` added higher level functions and commands
  - Interface to allocation policies
  - Creation/inspection of CPU & nodes bitmaps
  - Bind tasks to specific nodes
  - `numactl` to show parameters, set/get (default) options...
  - `numastat` displays kernel statistics on NUMA stuff
  - `migratepages` move pages of a process from nodes to other ones
- `numactl`, `libnuma-dev` and `libnuma1` packages on Debian

Bibliothèques mathématiques (I)

- Souvent bibliothèques constructeurs optimisées pour leur machines
  - Intel Math Kernel Library (MKL)
  - AMD Performance Library (⇝ Framewave libre)
- FFT : FFTW (*Fastest Fourier Transform in the West*, en C généré par du OCaml)...
- Beaucoup d'algèbre linéaire
  - BLAS (*Basic Linear Algebra Subprograms*) et PBLAS
  - LAPACK (*Linear Algebra PACKage*)
  - ScaLAPACK : version SPMD avec MPI
  - SuperLU : solution directe de gros systèmes creux
  - PETSc (*Portable, Extensible Toolkit for Scientific Computation*) : large spectre, au dessus de MPI

Processeur STI Cell (I)

- DOE/NNSA/LANL United States : Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband, 122400 cœurs, 2,345 MW, 1+ TFLOPS : TOP1 06/2008
- Démocratisation des machines en pannes : Sony PlayStation 3 avec Cell & 1 cœur en panne (?)
  - PS3 : bonne machine pour tester Cell à 400€ mais seulement 256 Mo (enseignement, tests)
  - Linux dans machine virtuelle
- PowerPC SMT 2 voies + 8 SPE PowerPC SIMD reliés par NoC
- Chaque SPE a 256 Ko mémoire *locale*, pas de MMU

Processeur STI Cell (II)

- Communication bas niveau : explicite par transferts DMA via `libspe`, synchronisations, boîtes aux lettres, contrôle des threads sur SPE...
  - Le retour des *overlays* ! Car programmes doivent aussi loger dans mémoire locale...
  - Attention aux plantages lors de phase de mise au point
  - Demande contrôle très fin chorégraphie des données (pavage, ≈*out-of-core*...)
- Environnement Linux bien intégré
- ∃ environnements de plus haut niveau
  - OpenMP avec simulation mémoire globale partagée (VSDM)
  - Paralléliseur automatique et VSDM

  Performances en fonction de l'application
- Simulateur intégré à Eclipse pour développer
- Processeur qui semble abandonné...

Outline

Classes

- Programmation objet qui utilise classes parallélisées ⇝ programmes parallèles !
- ∃ Versions parallélisées de la STL, souvent logiciel libre (PaSTel @ INRIA...)
- Boost contient aussi des choses liées au parallélisme (threads) et au calcul (uBLAS...)

  `http://en.wikipedia.org/wiki/Boost_library`
- Souvent le cas de bibliothèques métiers (bases de données, interfaces graphiques...)

Thread Building Blocks (TBB) (I)

`http://en.wikipedia.org/wiki/TBB` from Intel

- Template library (*à la* STL)
- Open and commercial Versions
- Algorithms (`for`, `reduce`, `pipeline`, `scan`...), containers, memory allocators, mutual exclusion, atomic operations, schedulers, profiling...
- Work stealing between tasks
- Orthogonal to OpenMP & MPI

Thread Building Blocks (TBB) (II)

```
1  class ApplyFoo {
2    float* const my_a;
   public:
4    ApplyFoo(float* a) : my_a(a) {};
     void operator() (const tbb::blocked_range<size_t>&r) const {
6      for (size_t i=r.begin(); i != r.end(); ++i)
         Foo (my_a[i]);
8    }
   }
10
   void ParallelApplyFoo(float a[], size_t n) {
12   tbb::parallel_for (
       tbb::blocked_range<size_t>(0,n),
14     ApplyFoo(a),
       tbb::auto_partitioner()
16   );
   }
```

- Need a deep code restructuring if an application is not in a STL spirit

ArBB (ex-Ct) : C/C++ for Throughput Computing (I)

- Array Building Blocks, fusion of Intel Ct language and technology from RapidMind
- C++ classes from Intel for its Terascale project
- Add « data collection » `TVEC` container
- Data-parallelism to deal easily with a lot of data
- Data-parallelism operations as in Nesl, PompC, HyperC... languages
  - Vector 1-to-1 functions
  - Collective communications: reductions, scan...
  - Permutations: scatter/gather, (un)packing, shift, partitions, butterfly...

ArBB (ex-Ct) : C/C++ for Throughput Computing

(II)

```
1  TVEC<F32> colorConvert(TVEC<F32> rchannel, TVEC<F32> gchannel,
                          TVEC<F32> bchannel, TVEC<F32> achannel,
3                         F32 a0, F32 a1, F32 a2, F32 a3) {
     return (rchannel*a0 + gchannel*a1 + bchannel*a2 + achannel*a3);
5  }

7  TVEC<F32> Convolve2D3x3(TVEC<F32> pixels, TVEC<F32> kernel) {
     TVEC<F32> respixels;
9    // directions[m][n] is a constant TVEC of size
     // with values {m-1, n-1}
11   respixels += shiftPermute(pixels, directions[0][0])*kernel[0][0];
     respixels += shiftPermute(pixels, directions[0][1])*kernel[0][1];
13   respixels += shiftPermute(pixels, directions[0][2])*kernel[0][2];
     respixels += shiftPermute(pixels, directions[1][0])*kernel[1][0];
15   respixels += pixels*kernel[1][1];
     respixels += shiftPermute(pixels, directions[1][2])*kernel[1][2];
17   respixels += shiftPermute(pixels, directions[2][0])*kernel[2][0];
     respixels += shiftPermute(pixels, directions[2][1])*kernel[2][1];
19   respixels += shiftPermute(pixels, directions[2][2])*kernel[2][2];
     return respixels
21 }
```

ArBB (ex-Ct) : C/C++ for Throughput Computing *(III)*

- Task graph to execute different data-parallel computations with some control parallelism
- Need some code restructuring to fit the data-parallel model

C++0x... 2011 (I)

- Parallelism and synchronization are mainstream ⇝ inclusion in last C++ version
- **#include** `<atomic>`
  - Atomic load, atomic store with different possible memory consistency models
  - Atomic side effect operators: `=`, `++`, `-=`...
  - Compare-exchange
  - Fetch-and-op
  - Some lock-free implementations
  - Memory fence
  - ...
- **#include** `<thread>`
  - Describe threads to be mapped one-to-one with OS threads
  - Creation
  - Join (wait for another thread to complete)
  - Get hardware concurrency hint

C++0x... 2011 (II)

- Sleep
- Yield

- `#include <mutex>`
  - Lock, recursive or not
  - Unlock
  - `try_lock()`, potentially with timeout

- `#include <condition_variable>`
  - Wait for a condition given by another thread
  - `wait()`
  - `notify_one()`, `notify_all()`

- `#include <future>`
  - Can get results from function execution (typically in another task)

Boost C++ (I)

- **#include** `<boost/thread.hpp>`
  - Pre-C++0x implementation
  - Contains in the same package also mutex, condition variables and future

  `http://www.quantnet.com/cplusplus-multithreading-boost`
- **#include** `<boost/mpi.hpp>`
  - C++ wrapping of Message Passing Interface
  - Better integration with STL spirit than native MPI C++ support

Outline

Profiling

- Use sampling techniques, instrumentation and/or hardware performance counters
  - If global, need some kernel support...
- `gprof` open source
- PAPItools open source access to hardware counters
- IBM HPCS toolkit
- AMD CodeAnalyst
- Intel VTune $\in$ Parallel Studio
- Intel Thread Profiler (included in VTune)
- Open|SpeedShop
- oprofile
- kcachegrind, callgrind... use virtual machine techniques for profiling `http://valgrind.org`
- Dtrace in Solaris to profile the kernel itself (cf. the « Shouting in the datacenter » great video)

gprof (I)

« *gprof: A Call Graph Execution Profiler* » S. Graham, P. Kessler, M. McKusick; *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, SIGPLAN Notices*, Vol. 17, No 6, pp. 120-126, June 1982.

- Instrument the code to trace execution at runtime
  - Code instrumentation by the compiler with `-pg` option
  - `gprof` analyzes post-mortem `gmon.out` file + executable
- Modify the execution: increment counters, record callees... ⇝
  Add instructions and modify execution with cache effects 
- Use interrupts for statistical time measurement
  - Analyse the stack where the interrupt occurred. Cf for example `backtrace()` from GNU libc
  - Quantification of the time
  -  need long time for precision
  - Uncommon small events can go unnoticed
-  Cumulated for all the threads

gprof (II)

- 3 kinds of output
  - Flat profile output
    - Time spent in each function
    - How much time each function is called
  - Call graph output
    - For each function, which functions it called and how many times
    - How much time spent it each of these called functions
  - Annotated source listing output
    - Listing of the program with the number of times each line is called
- `main` should be compiled with `-pg` too...

OProfile

- Eclipse plugin for OProfile
  `http://www.eclipse.org/linuxtools/projectPages/oprofile`
- Using a JIT is challenging: generate new code on the fly, unknown *a priori* from the system...
  - When JITing, provide information about the new code to the profiler: new code at a given address, symbol name, debug info...
  - Need to develop a JIT agent for OProfile
  - OProfile come already with a Java Virtual Machine Toolkit Interface (JVMTI) library, `libjvmti_oprofile.so`

« *Tuning Programs with OProfile* », William E. COHEN, WIDE OPEN MAGAZINE, 2004, N. 1, pp. 53–62.
`http://people.redhat.com/wcohen/Oprofile.pdf`

Debugging and thread checking

- Runtime checking of data access properties
- Helgrind `http://valgrind.org`
  - Open source tool from the Valgrind suite (dynamic binary instrumentation) for Unix
  - `valgrind --tool=helgrind ./pi_SPMD`
- Intel Thread checker
  - Binary instrumentation
  - GUI front-end for Windows & Linux
-  *Dynamic* tools verify *only the executed part* of the code...
   Good coverage tests needed here...

Translators

- Ease programmer's life
  - Vectorizer: sequential program ⇝ program with vector instructions
  - Parallelizer: sequential program ⇝ program for parallel machine
- Avoid explicit parallelism, but often not very efficient
  - Hard in the general case, many C issues
  - Unreachable for object oriented such as C++
  - Need some help ⇝ directives (vendors, HPF, OpenMP...)
- Current trends
  - Recycling old tricks from auto-parallelizers in all the tools: GCC, icc, Cuda, Brook+, HMPP...
  - Mix automation with `#pragma`tism

Passage à l'échelle (I)

- Réseaux de la recherche rapide (RENATER, VTHD++, ABILENE,...)
- Mondialisation des infrastructures
- Exploiter les « jachères de calcul » disponibles
- Protocoles plus sexy et interopérables (XML)
- Mode des Web services

⇝ Grilles de calcul

-  Ne marche que pour des applications tolérant la latence...

Stockage pair à pair

- Besoins de stocker et partager des données sur plusieurs sites
- Recherche et indexation
- Multiprotocole (HTTP vu comme un protocole de transport qui traverse les parefeux...)
- Identifiants uniques indépendant des protocoles ou adresses (NAT,...)
- Construction d'une infrastructure entre pairs au dessus du réseau réel
  - Nœuds de routage
  - Nœuds de stockage
  - Mandataires (*proxy*)
- Exemple : JXTA de Sun en Java, OceanStore,...
-  Ne marche que pour des applications tolérant la latence...

Calcul distribué mondial en client-serveur

Majorité des ordinateurs de la planète ne font qu'attendre l'utilisateur... ☺⇝ Des EFLOPS potentiels gratuits

- SETI@home : remplace les économiseurs (useurs ?) d'écran pour rechercher de signaux extra-terrestres
- Décrypthon : bio-informatique
- Concours de cassage d'algorithmes de cryptographie
-  Ne marche que pour des applications tolérant la latence...

Grilles de calcul

- Idée offrir des ressources de calcul semblable au service d'électricité
- Faire des requêtes de ressources à un supercalculateur virtuel
- Mutualisation des ressources pour utiliser des puissances de calcul inabordable autrement
- Exemple : Globus2 vu comme un service Web
- Problèmes : latences considérables, débits souvent faibles, identifications globales (PKI),...
- Besoin d'une algorithmique adaptée à très gros grain : couplage de code différents (transport + modèle biologique,...)

-  Ne marche que pour des applications tolérant la latence...

⇝ + virtualisation: cloud computing

Outline

We need software tools

- HPC Project needs tools for its hardware accelerators (*Wild Nodes* from *Wild Systems*) and to parallelize, port & optimize customer applications
- Application development: long-term business ⇝ long-term commitment in a tool that needs to survive to (too fast) technology change

Not reinventing the wheel... No NIH syndrome please

Want to create your own tool?

- House-keeping and infrastructure in a compiler is a huge task
- Unreasonable to begin yet another new compiler project...
- Many academic Open Source projects are available...
- ...But customers need products ☺
- ⇝ Integrate your ideas and developments in existing project
- ...or buy one if you can afford (ST with PGI...) ☺
- Some projects to consider
  - Old projects: gcc, PIPS... and many dead ones (SUIF...)
  - But new ones appear too: LLVM, RoseCompiler, Cetus...

Par4All

- ⇝ Funding an initiative to industrialize Open Source tools
- PIPS is the first project to enter the Par4All initiative

`http://www.par4all.org`

PIPS (I)

- PIPS (Interprocedural Parallelizer of Scientific Programs): Open Source project from Mines ParisTech... 23-year old! ☺
- Funded by many people (French DoD, Industry & Research Departments, University, CEA, IFP, Onera, ANR (French NSF), European projects, regional research clusters...)
- One of the project that coined polytope model-based compilation
- ≈ 456 KLOC according to David A. Wheeler's `SLOCCount`
- ... but modular and sensible approach to pass through the years
  - ≈300 phases (parsers, analyzers, transformations, optimizers, parallelizers, code generators, pretty-printers...) that can be combined for the right purpose
  - Polytope lattice (sparse linear algebra) used for semantics analysis, transformations, code generation... to deal with big programs, not only loop-nests

PIPS (II)

- NewGen object description language for language-agnostic automatic generation of methods, persistence, object introspection, visitors, accessors, constructors, XML marshaling for interfacing with external tools...
- Interprocedural *à la* `make` engine to chain the phases as needed. Lazy construction of resources
- On-going efforts to extend the semantics analysis for C

- Around 15 programmers currently developing in PIPS (Mines ParisTech, HPC Project, IT SudParis, TÉLÉCOM Bretagne, RPI) with public `svn`, Trac, `git`, mailing lists, IRC, Plone, Skype... and use it for many projects
- But still...
  - Huge need of documentation (even if PIPS uses literate programming...)
  - Need of industrialization
  - Need further communication to increase community size

Current PIPS usage

- Automatic parallelization (Par4All C & Fortran to OpenMP)
- Distributed memory computing with OpenMP-to-MPI translation [STEP project]
- Generic vectorization for SIMD instructions (SSE, VMX, Neon, CUDA, OpenCL...) (SAC project) [SCALOPES]
- Parallelization for embedded systems [SCALOPES, SMECY]
- Compilation for hardware accelerators (Ter@PIX, SPoC, SIMD, FPGA...) [FREIA, SCALOPES]
- High-level hardware accelerators synthesis generation for FPGA [PHRASE, CoMap]
- Reverse engineering & decompiler (reconstruction from binary to C)
- Genetic algorithm-based optimization [Luxembourg university+TB]
- Code instrumentation for performance measures
- GPU with CUDA & OpenCL [TransMedi@, FREIA, OpenGPU]

p4a in a nutshell (I)

Parallelisation

`p4a matmul.f`

generates an OpenMP program in `matmul.p4a.f`

```
1  !$omp parallel do private(I, K, X)
2  C multiply the two square matrices of ones
         DO J = 1, N
4  !$omp parallel do private(K, X)
            DO I = 1, N
6              X = 0
  !$omp parallel do reduction(+:X)
8              DO K = 1, N
                 X = X+A(I,K)*B(K,J)
10             ENDDO
  !$omp end parallel do
12             C(I,J) = X
            ENDDO
14 !$omp end parallel do
         ENDDO
16 !$omp end parallel do
```

p4a in a nutshell (II)

Parallelisation with compilation

```
p4a matmul.f -o matmul
```

generates an OpenMP program `matmul.p4a.f` that is compiled with `gcc` into `matmul`

CUDA generation with compilation

```
p4a --cuda saxpy.c -o s
```

generates a CUDA program that is compiled with `nvcc`

Parallelization options

`p4a [options] files`

- `--openmp` generates OpenMP output. This is the default behaviour
- `--cuda` generates CUDA output
- `--accel --openmp` generates OpenMP output that simulates the CUDA behaviour

Automatic parallelization

Most fundamental for a parallel execution

Finding parallelism!

Several parallelization algorithms are available in PIPS

- For example classical Allen & Kennedy use loop distribution more vector-oriented than kernel-oriented  (or need later loop-fusion)
- Coarse grain parallelization based on the independence of array regions used by different loop iterations
  - Currently used because generates GPU-friendly coarse-grain parallelism
  - Accept complex control code without *if-conversion*

Outlining (I)

Parallel code ⇝ Kernel code on GPU

- Need to extract parallel source code into kernel source code: outlining of parallel loop-nests
- Before:

```
1     for(i = 1;i <= 499; i++)
2       for(j = 1;j <= 499; j++) {
          save[i][j] = 0.25*(space[i - 1][j] + space[i + 1][j]
4                            + space[i][j - 1] + space[i][j + 1]);
        }
```

Outlining (II)

- After:

```
1  p4a_kernel_launcher_0(space, save);
   [...]
3  void p4a_kernel_launcher_0(float_t space[SIZE][SIZE],
                              float_t save[SIZE][SIZE]) {
5    for(i = 1; i <= 499; i += 1)
       for(j = 1; j <= 499; j += 1)
7        p4a_kernel_0(i, j, save, space);
   }
9  [...]
   void p4a_kernel_0(float_t space[SIZE][SIZE],
11                   float_t save[SIZE][SIZE],
                     int i,
13                   int j) {
     save[i][j] = 0.25*(space[i-1][j]+space[i+1][j]
15                      +space[i][j-1]+space[i][j+1]);
   }
```

From array regions to GPU memory allocation (I)

- Memory accesses are summed up for each statement as *regions* for array accesses: integer polytope lattice
- There are regions for write access and regions for read access
- The regions can be exact if PIPS can prove that only these points are accessed, or they can be inexact, if PIPS can only find an over-approximation of what is really accessed

From array regions to GPU memory allocation (II)

Example

```
1 for(i = 0; i <= n-1; i += 1)
2   for(j = i; j <= n-1; j += 1)
      h_A[i][j] = 1;
```

can be decorated by PIPS with write array regions as:

```
1 //  <h_A[PHI1][PHI2]-W-EXACT-{0<=PHI1, PHI2+1<=n, PHI1<=PHI2}>
    for(i = 0; i <= n-1; i += 1)
3 //  <h_A[PHI1][PHI2]-W-EXACT-{PHI1==i, i<=PHI2, PHI2+1<=n, 0<=i}>
      for(j = i; j <= n-1; j += 1)
5 //  <h_A[PHI1][PHI2]-W-EXACT-{PHI1==i, PHI2==j, 0<=i, i<=j, 1+j<=n}>
        h_A[i][j] = 1;
```

- These read/write regions for a kernel are used to allocate with a `cudaMalloc()` in the host code the memory used inside a kernel and to deallocate it later with a `cudaFree()`

Communication generation (I)

Conservative approach to generate communications

- Associate any GPU memory allocation with a copy-in to keep its value in sync with the host code
- Associate any GPU memory deallocation with a copy-out to keep the host code in sync with the updated values on the GPU

-  But a kernel could use an array as a local (private) array
- ...PIPS does have many privatization phases ☺
-  But a kernel could initialize an array, or use the initial values without writing into it or use/touch only a part of it or...

Communication generation (II)

More subtle approach

PIPS gives 2 very interesting region types for this purpose

- In-region abstracts what really needed by a statement
- Out-region abstracts what really produced by a statement to be used later elsewhere

- In-Out regions can directly be translated with CUDA into
  - copy-in

```
1  cudaMemcpy(accel_address, host_address,
2             size, cudaMemcpyHostToDevice)
```

  - copy-out

```
1  cudaMemcpy(host_address, accel_address,
2             size, cudaMemcpyDeviceToHost)
```

Loop normalization

- Hardware accelerators use fixed iteration space (thread index starting from 0...)
- Parallel loops: more general iteration space
- Loop normalization

Before

```
1 for(i = 1;i < SIZE - 1; i++)
2   for(j = 1;j < SIZE - 1; j++) {
      save[i][j] = 0.25*(space[i - 1][j] + space[i + 1][j]
4                        + space[i][j - 1] + space[i][j + 1]);
    }
```

After

```
1 for(i = 0;i < SIZE - 2; i++)
    for(j = 0;j < SIZE - 2; j++) {
3     save[i+1][j+1] = 0.25*(space[i][j + 1] + space[i + 2][j + 1]
                           + space[i + 1][j] + space[i + 1][j + 2]);
5   }
```

From preconditions to iteration clamping (I)

- Parallel loop nests are compiled into a CUDA kernel wrapper launch
- The kernel wrapper itself gets its virtual processor index with some `blockIdx.x*blockDim.x + threadIdx.x`
- Since only full blocks of threads are executed, if the number of iterations in a given dimension is not a multiple of the `blockDim`, there are incomplete blocks ☹
- An incomplete block means that some index overrun occurs if all the threads of the block are executed 

From preconditions to iteration clamping (II)

- So we need to generate code such as

```
1  void p4a_kernel_wrapper_0(int k, int l,...)
2  {
     k = blockIdx.x*blockDim.x + threadIdx.x;
4    l = blockIdx.y*blockDim.y + threadIdx.y;
     if (k >= 0 && k <= M - 1 && l >= 0 && l <= M - 1)
6      kernel(k, l, ...);
   }
```

But how to insert these guards?

- The good news is that PIPS owns *preconditions* that are predicates on integer variables. Preconditions at entry of the kernel are:

```
1  //  P(i,j,k,l) {0<=k, k<=63, 0<=l, l<=63}
```

- Guard $\equiv$ directly translation in C of preconditions on loop indices that are GPU thread indices

Complexity analysis (I)

- Launching a GPU kernel is very expensive, launching OpenMP Parallel section is expensive too
  - so we need to launch only kernels with a significant speed-up (launching overhead, memory CPU-GPU copy overhead...)
- Some systems use `#pragma` to give a go/no-go information to parallel execution

```
1  #pragma omp parallel if(size>100)
```

- ∃ phase in PIPS to symbolically estimate complexity of statements
- Based on preconditions
- Use a SuperSparc2 model from the '90s... ☺
- Can be changed, but precise enough to have a coarse go/no-go information
- To be refined: use memory usage complexity to have information about memory reuse (even a big kernel could be more efficient on a CPU if there is a good cache use)

Optimized reduction generation

- Reduction are common patterns that need special care to be correctly parallelized

$$s = \sum_{i=0}^{N} x_i$$

- Reduction detection already implemented in PIPS
- Generate **#pragma** `omp reduce` in Par4All

Communication optimization

- Naive approach : load/compute/store
- Useless communications if a data on GPU is not used on host between 2 kernels... ☹
- ⇝ Use static interprocedural data-flow communications
  - Fuse various GPU arrays : remove GPU (de)allocation
  - Remove redundant communications

⇝ New `p4a --com-optimization` option

Loop fusion

- Programs ≡ often a succession of (parallel) loops
- Can be interesting to fuse loops together
  - Important for array-oriented languages: Fortran 95, Scilab, C++ parallel class...
  - Factorize control : one loop with bigger content
    - More important for heterogeneous accelerators: reduce kernel launch time
    - May avoid memory round trip
    - May cache recycling
- Use dependence graph, regions... to figure out when to fuse
- Sensible parallel promotion of scalar code to reduce parallelism interruption still to be implemented

Dead code elimination

- ∃ useless part of source in real applications
  - History, big-ball-of-mud...
  - Different configuration of a same program according to input values, preprocessor...
  -  Dead code ≡ legal parallel code with PIPS semantics analyses ☺Confusing... ☹
- Very common in automatically generated code
  - Easier to generate systematic code
  - Rely on other phases to do the cleaning
- In PIPS
  - Use-def elimination from IO ( lousy benchmarks...)
  - Partial evaluation, constant propagation
  - Loops & test simplifications according to preconditions on scalar variables

Fortran to C-based GPU languages

- Fortran 77 parser available in PIPS
- CUDA & OpenCL are C++/C99 with some restrictions on the GPU-executed parts
- Need a Fortran to C translator (`f2c`...)?
- Only one internal representation is used in PIPS
  - Use the Fortran parser
  - Use the... C pretty-printer!
- But the IO Fortran library is complex to use... and to translate
  - If you have IO instructions in a Fortran loop-nest, it is not parallelized anyway because of sequential side effects ☺
  - So keep the Fortran output everywhere but in the parallel CUDA kernels
  - Apply a memory access transposition phase `a(i,j)` ⇝ `a[j-1][i-1]` inside the kernels to be pretty-printed as C
- Compile and link C GPU kernel parts + Fortran main parts
- Quite harder than expected... Use Fortran 2003 for C interfaces...

Outline

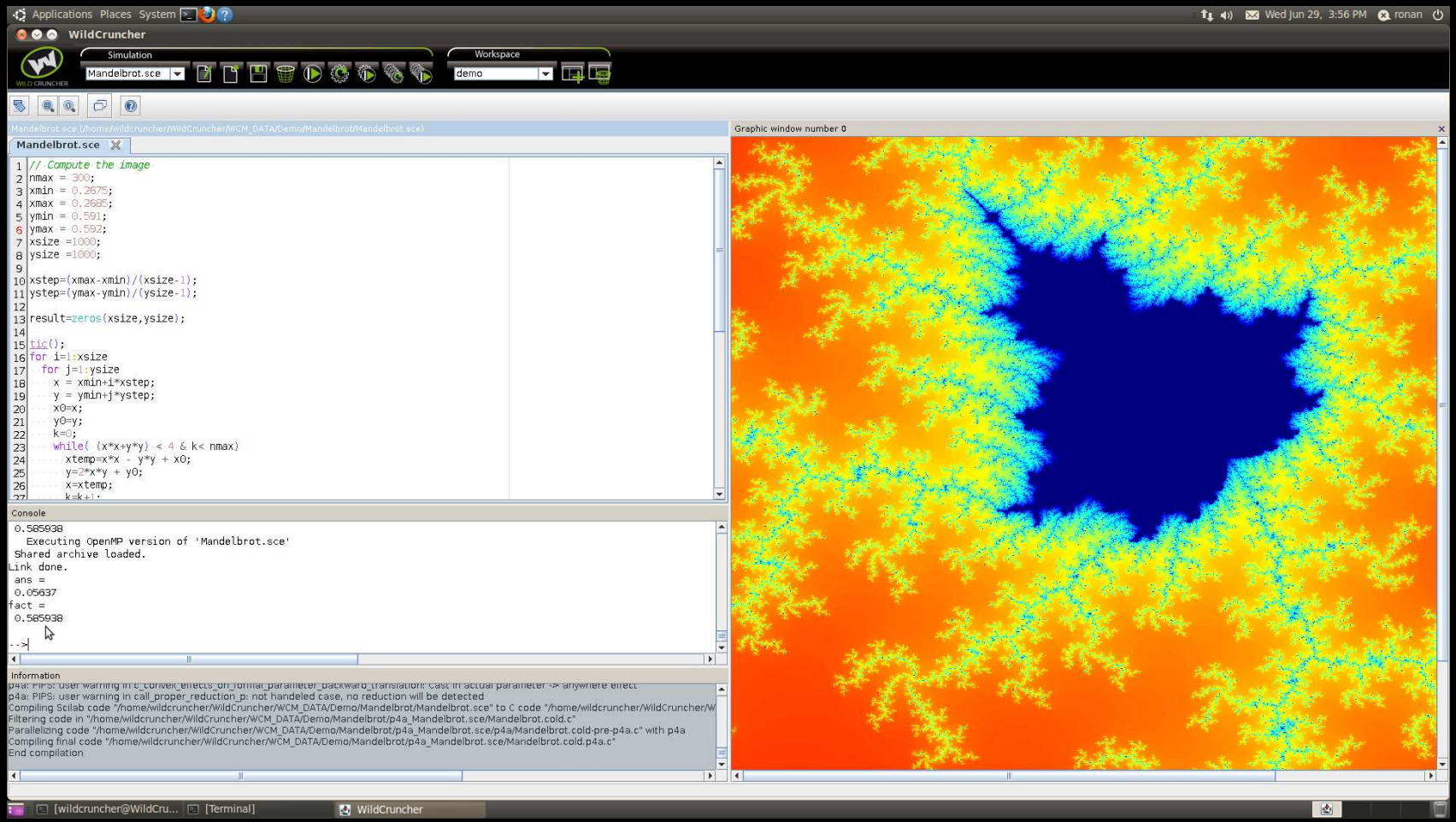
Scilab language

- Interpreted scientific language widely used like Matlab
- Free software
- Roots in free version of Matlab from the 80's
- Dynamic typing (scalars, vectors, (hyper)matrices, strings...)
- Many scientific functions, graphics...
- Double precision everywhere, even for loop indices (now)
- Slow because everything decided at runtime, garbage collecting
  - Implicit loops around each vector expression
    - Huge memory bandwidth used
    - Cache thrashing
    - Redundant control flow
- Strong commitment to develop Scilab through Scilab Enterprise, backed by a big user community, INRIA...
- HPC Project WildNode appliance with Scilab parallelization
- Reuse Par4All infrastructure to parallelize the code

Scilab & Matlab (I)

- Scilab/Matlab input : *sequential* or array syntax
- Compilation to C code
- Parallelization of the generated C code
- Type inference to guess (crazy ☹) semantics
  - Heuristic: first encountered type is forever
- Speedup $>$ 1000 ☺
- Wild Cruncher: *x*86+GPU appliance with nice interface
  - Scilab — mathematical model & simulation
  - Par4All — automatic parallelization
  - //Geometry — polynomial-based 3D rendering & modelling
- Versions to compile to other platforms (fixed-point DSP...)

Wild Cruncher — Scilab parallelization



Outline

Hyantes (I)

- Geographical application: library to compute neighbourhood population potential with scale control
- Example given in `par4all.org` distribution
- WildNode with 2 Intel Xeon X5670 @ 2.93GHz (12 cores) and a nVidia Tesla C2050 (Fermi), Linux/Ubuntu 10.04, gcc 4.4.3, CUDA 3.1
  - Sequential execution time on CPU: 30.355s
  - OpenMP parallel execution time on CPUs: 3.859s, speed-up: 7.87
  - CUDA parallel execution time on GPU: 0.441s, speed-up: 68.8
- With single precision on a HP EliteBook 8730w laptop (with an Intel Core2 Extreme Q9300 @ 2.53GHz (4 cores) and a nVidia GPU Quadro FX 3700M (16 multiprocessors, 128 cores, architecture 1.1)) with Linux/Debian/sid, gcc 4.4.5, CUDA 3.1:
  - Sequential execution time on CPU: 34.7s
  - OpenMP parallel execution time on CPUs: 13.7s, speed-up: 2.53
  - OpenMP emulation of GPU on CPUs: 9.7s, speed-up: 3.6

Hyantes (II)

- CUDA parallel execution time on GPU: 1.57s, speed-up: 24.2

Original main C kernel:

```
1  void run(data_t xmin, data_t ymin, data_t xmax, data_t ymax, data_t step, data_t range,
2    town pt[rangex][rangey], town t[nb])
3  {
4      size_t i,j,k;
5
6      fprintf(stderr,"begin computation ...\n");
7
8      for(i=0;i<rangex;i++)
9          for(j=0;j<rangey;j++) {
10             pt[i][j].latitude =(xmin+step*i)*180/M_PI;
11             pt[i][j].longitude =(ymin+step*j)*180/M_PI;
12             pt[i][j].stock =0.;
13             for(k=0;k<nb;k++) {
14                 data_t tmp = 6368.* acos(cos(xmin+step*i)*cos( t[k].latitude )
15                      * cos((ymin+step*j)-t[k].longitude)
16                      + sin(xmin+step*i)*sin(t[k].latitude));
17                 if( tmp < range )
18                     pt[i][j].stock += t[k].stock  / (1 + tmp) ;
19             }
20         }
21     fprintf(stderr,"end computation ...\n");
22 }
```

Hyantes (III)

OpenMP code:

```
1  void run(data_t xmin, data_t ymin, data_t xmax, data_t ymax, data_t step, d
2  {
3     size_t i, j, k;
4
5     fprintf(stderr, "begin computation ...\n");
6
7  #pragma omp parallel for private(k, j)
8     for(i = 0; i <= 289; i += 1)
9        for(j = 0; j <= 298; j += 1) {
10          pt[i][j].latitude = (xmin+step*i)*180/3.14159265358979323846;
11          pt[i][j].longitude = (ymin+step*j)*180/3.14159265358979323846;
12          pt[i][j].stock = 0.;
13          for(k = 0; k <= 2877; k += 1) {
14             data_t tmp = 6368.*acos(cos(xmin+step*i)*cos(t[k].latitude)*cos
15             if (tmp<range)
16                pt[i][j].stock += t[k].stock/(1+tmp);
17          }
18       }
19    fprintf(stderr, "end computation ...\n");
20 }
21 void display(town pt[290][299])
22 {
```

Hyantes (IV)

```
    size_t i, j;
    for(i = 0; i <= 289; i += 1) {
        for(j = 0; j <= 298; j += 1)
            printf("%lf %lf %lf\n", pt[i][j].latitude, pt[i][j].longitude, pt[i
        printf("\n");
    }
}
```

Hyantes (V)

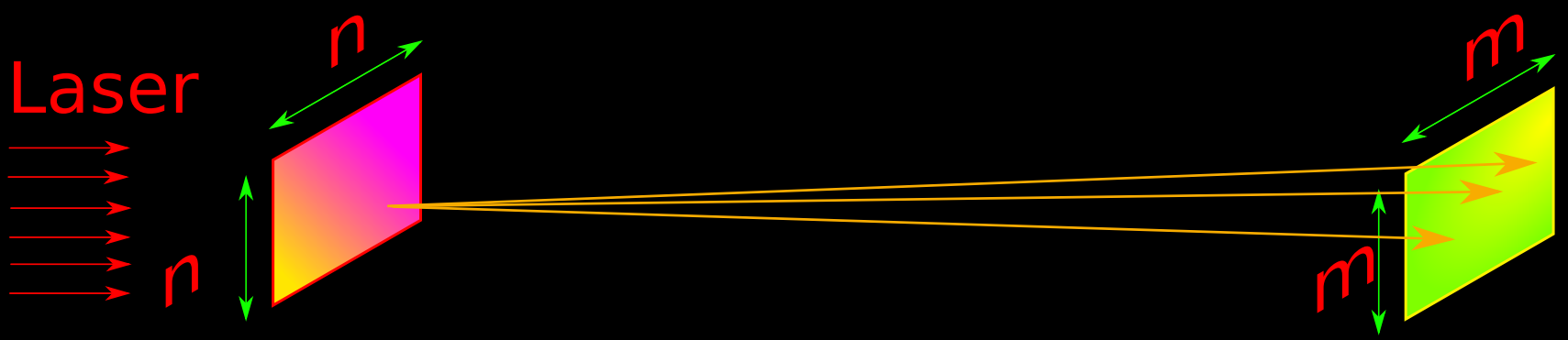
Generated GPU code:

```
1  void run(data_t xmin, data_t ymin, data_t xmax, data_t ymax, data_t step, data_t range,
     town pt[290][299], town t[2878])
3  {
     size_t i, j, k;
5    //PIPS generated variable
     town (*P_0)[2878] = (town (*)[2878]) 0, (*P_1)[290][299] = (town (*)[290][299]) 0;
7
     fprintf(stderr, "begin computation ...\n");
9    P4A_accel_malloc(&P_1, sizeof(town[290][299])-1+1);
     P4A_accel_malloc(&P_0, sizeof(town[2878])-1+1);
11   P4A_copy_to_accel(pt, *P_1, sizeof(town[290][299])-1+1);
     P4A_copy_to_accel(t, *P_0, sizeof(town[2878])-1+1);
13
     p4a_kernel_launcher_0(*P_1, range, step, *P_0, xmin, ymin);
15   P4A_copy_from_accel(pt, *P_1, sizeof(town[290][299])-1+1);
     P4A_accel_free(*P_1);
17   P4A_accel_free(*P_0);
     fprintf(stderr, "end computation ...\n");
19 }

21 void p4a_kernel_launcher_0(town pt[290][299], data_t range, data_t step, town t[2878],
     data_t xmin, data_t ymin)
23 {
     //PIPS generated variable
25   size_t i, j, k;
     P4A_call_accel_kernel_2d(p4a_kernel_wrapper_0, 290,299, i, j, pt, range,
27                            step, t, xmin, ymin);
29 }

P4A_accel_kernel_wrapper void p4a_kernel_wrapper_0(size_t i, size_t j, town pt[290][299],
```

Hyantes (VI)

```
31    data_t range, data_t step, town t[2878], data_t xmin, data_t ymin)
{
33    // Index has been replaced by P4A_vp_0:
      i = P4A_vp_0;
35    // Index has been replaced by P4A_vp_1:
      j = P4A_vp_1;
37    // Loop nest P4A end
      p4a_kernel_0(i, j, &pt[0][0], range, step, &t[0], xmin, ymin);
39  }

41  P4A_accel_kernel void p4a_kernel_0(size_t i, size_t j, town *pt, data_t range,
      data_t step, town *t, data_t xmin, data_t ymin)
43  {
      //PIPS generated variable
45    size_t k;
      // Loop nest P4A end
47    if (i<=289&&j<=298) {
        pt[299*i+j].latitude = (xmin+step*i)*180/3.14159265358979323846;
49      pt[299*i+j].longitude = (ymin+step*j)*180/3.14159265358979323846;
        pt[299*i+j].stock = 0.;
51      for(k = 0; k <= 2877; k += 1) {
          data_t tmp = 6368.*acos(cos(xmin+step*i)*cos((*(t+k)).latitude)*cos(ymin+step*j
53              -(*(t+k)).longitude)+sin(xmin+step*i)*sin((*(t+k)).latitude));
          if (tmp<range)
55          pt[299*i+j].stock += t[k].stock/(1+tmp);
        }
57    }
  }
```

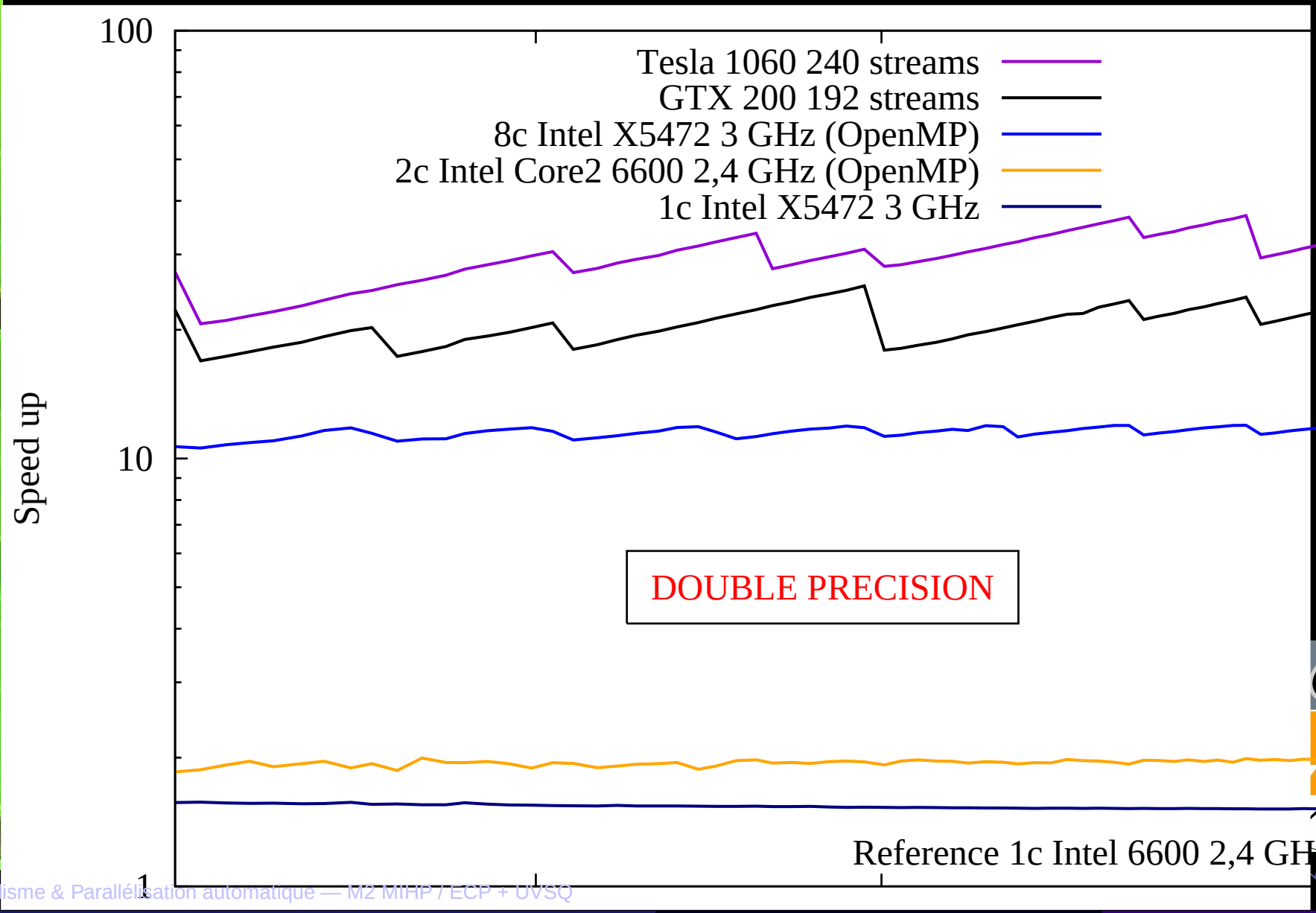
Results on a customer application



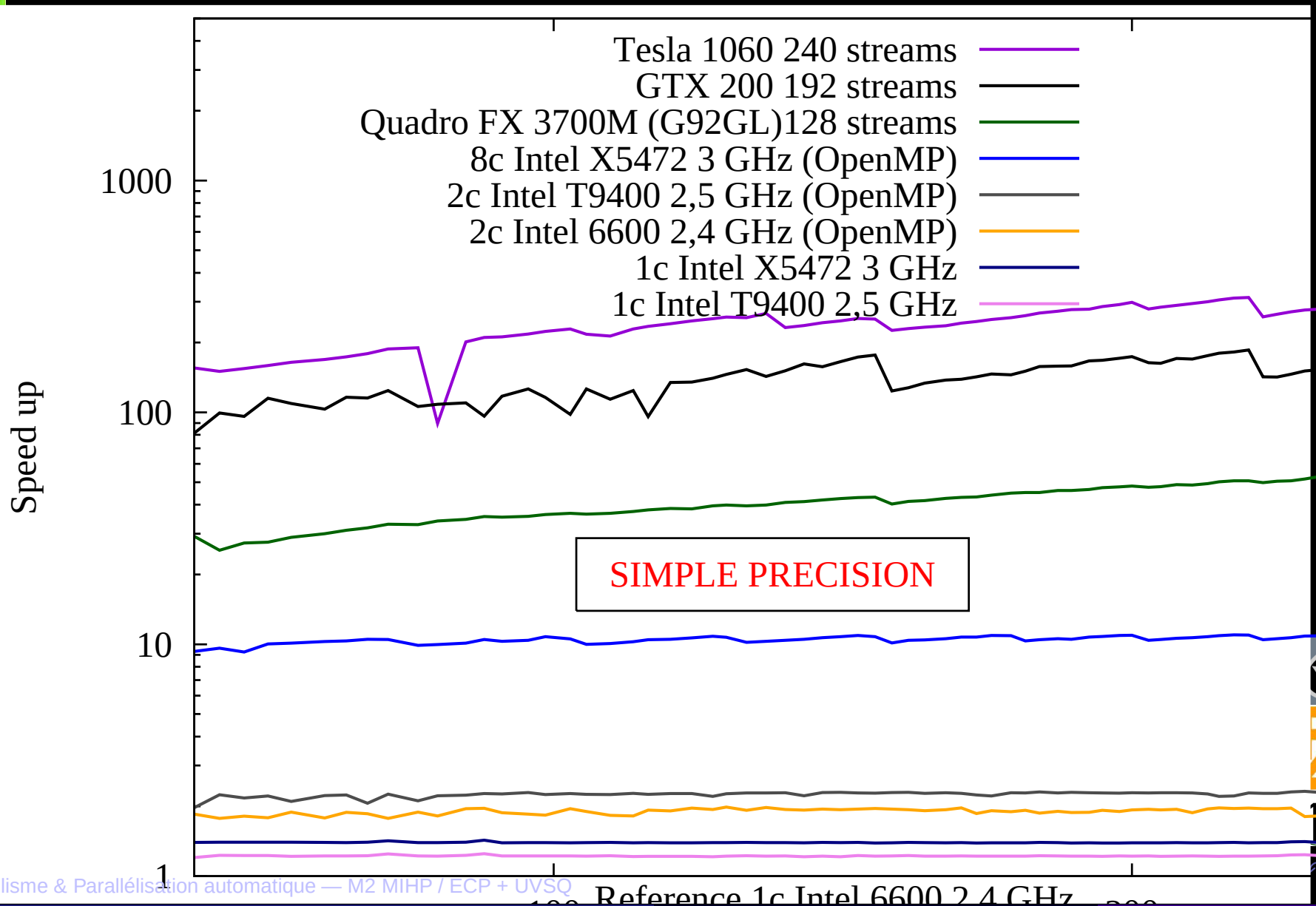
- Holotetrix's primary activities are the design, fabrication and commercialization of prototype diffractive optical elements (DOE) and micro-optics for diverse industrial applications such as LED illumination, laser beam shaping, wavefront analyzers, etc.
- Hologram verification with direct Fresnel simulation
- Program in C
- Parallelized with
  - Par4All CUDA and CUDA 2.3, Linux Ubuntu x86-64
  - Par4All OpenMP, gcc 4.3, Linux Ubuntu x86-64
- Reference: Intel Core2 6600 @ 2.40GHz

`http://www.holotetrix.com`

Comparative performance



Keep it simple (precision)



Stars-PM

- *Particle-Mesh* N-body cosmological simulation
- C code from Observatoire Astronomique de Strasbourg
- Use FFT 3D
- Example given in `par4all.org` distribution

Stars-PM time step

```
1  void iteration(coord pos[NP][NP][NP],
2                 coord vel[NP][NP][NP],
                  float dens[NP][NP][NP],
4                 int data[NP][NP][NP],
                  int histo[NP][NP][NP]) {
6    /* Découpe l'espace tridimensionnel
        selon une grille régulière */
8  discretisation(pos, data);
     /* Calcul de la densité sur la grille */
10 histogram(data, histo);
     /* Calcul du potentiel sur la grille
12      dans l'espace de Fourier */
   potential(histo, dens);
14   /* Calcul dans chaque dimension de la force
        et application à la vitesse des particules */
16 forcex(dens, force);
   updatevel(vel, force, data, 0, dt);
18 forcey(dens, force);
   updatevel(vel, force, data, 1, dt);
20 forcez(dens, force);
   updatevel(vel, force, data, 2, dt);
22   /* Déplacement des particules */
   updatepos(pos, vel);
24 }
```

Stars-PM & Jacobi results with `p4a 1.1` *(I)*

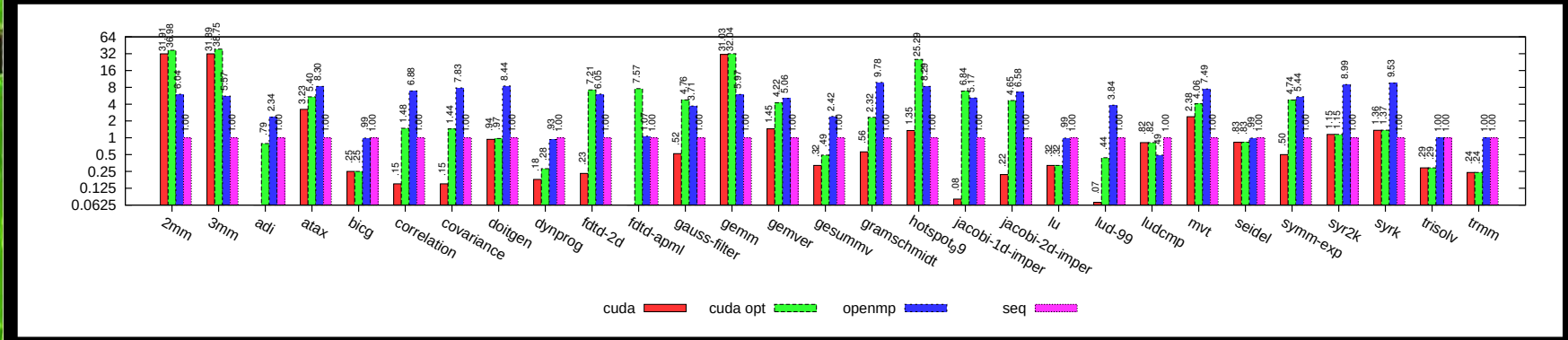
- 2 Xeon Nehalem X5670 (12 cores @ 2,93 GHz)
- 1 GPU nVidia Tesla C2050
- Automatic call to CuFFT instead FFTW
- 150 iterations of Stars-PM

Temps d'exécution	`p4a`	Simulation Cosmo. 32^3	64^3	128^3	Jacobi
Séquentiel	(`gcc -O3`)	0,68	6,30	98,4	24,5
OpenMP 6 *threads*	`--openmp`	0,16	1,28	16,6	13,8
CUDA base	`--cuda`	0,88	5,21	31,4	67,7
Comm. optimisées	`--cuda --com-opt.`	0,20	1,17	8,9	6,5
Optimisation manuelles	(`gcc -O3`)	0,05	0,26	1,7	

Current limitation for Stars-PM with `p4a`: histogram is not parallelized...

Benchmark results

With Par4All 1.2, CUDA 4.0, WildNode 2 Xeon Nehalem X5670 (12 cores @ 2.93 GHz) with nVidia C2050



From `par4all.org` distribution, in `examples/Benchmarks`

Outline

Coding rules

- Automatic parallelization is not magic
- Use abstract interpretation to « understand » programs
- Undecidable in the generic case ($\approx$ halting problem)
- Automatic parallelization: big deception from the past... ☹
- Quite easier for well written programs ☺
- Develop a coding rule manual to help parallelization and... sequential quality!
  - Avoid useless pointers
  - Take advantage of C99 (arrays of non static size...)
    
    Do not forget `ulimit -s unlimited` in the shell to remove limitation on the stack size
  - Use higher-level C, do not linearize arrays...
  - Organize execution in cleaner loops expressing better parallelism
  - ...
- Prototype of coding rules report on-line on `par4all.org`

Profiling

- Use profiling to figure out where the hot-spots are
- Verify hot-spots are well parallelized by Par4All
- If not, rewrite hot-spots cleanly so they can be parallelized by Par4All
- Isolate code to be parallelized into functions with clean interfaces
- If not parallelized: bug report and/or subcontract! ☺

Bad/good C programming example

Dynamically allocated 2-D array with dynamic size

- Bad:

```
1  int n, m;
2  double * t =
     malloc(sizeof(double)*n*m);
4  for(int i = 0; i < n; i++)
     for(int j = 0; j < m; m++)
6      t[i*n + j] = i + j;
   free(t);
```

- Good:

```
1  double (*t)[n][m] =
     malloc(sizeof(double (*)[n][m]));
3  for(int i = 0; i < n; i++)
     for(int j = 0; j < m; m++)
5      (*t)[i][j] = i + j;
   free(t);
```

- Good:

```
1  {
2    double t[n][m];
     for(int i = 0; i < n; i++)
4      for(int j = 0; j < m; m++)
         t[i][j] = i + j;
6  }
```

↝ Before parallelization, good sequential programming...

Take advantage of C99 (I)

Write C99impler C99ode, easier to parallelize automatiC99ally...

- BaC99k to C99imple C99ings
- Avoiding using C99umbersome old C C99onstruC99ts C99an lead to C99leaner C99ode, more effiC99ient and more parallelizable C99ode (with Par4All...)
- C99 adds C99ympaC99etiC99 features that are unfortunately not well known:
  - MultidimenC99ional arrays with non-statiC99 size (VLA)
    - Avoid `malloc()` spam and useless pointer C99onstruC99ions
    - You C99an have arrays with a dynamiC99 size in funC99tion parameters (as in Fortran)
    - Avoid useless linearizing C99omputaC99ions (`a[i][j]` instead of `a[i+n*j]`...)
      Avoid non-affine constructs that are hard to analyze for parallelization, communications...
    - Avoid most of `alloca()`

Take advantage of C99 *(II)*

- TLS (*Thread-Local Storage*) ExtenC$_{99}$ion C$_{99}$an express independent storage
- C$_{99}$omplex numbers and booleans avoid further struC$_{99}$tures or enums

↝ C$_{99}$ is good for you!

- C2011 is out and VLA are now only optional... ☹

Clean programming for Par4All (I)

- Pointers and global variables are often issues with parallelization
- OpenMP parallelization is more tolerant, but compilation for heterogeneous computing need more difficult communication analysis
- Typical pattern to hide nastiness under the carpet:

```
1  double global_array[N][M];
2  void some_gory_function(float *array, int size) {
     int auto_array[size];
4    double (*heap_array)[size][size] = malloc(sizeof(double (*)[size][siz
     float (*cleaner_array)[size] = (float (*)[size]) array;
6
     clean_function(size, auto_array, *heap_array, *cleaner_array, global_
8  }

10 // Here all arrays are C99-style now!
   void clean_function(int size, int auto_array[size],
12                     double heap_array[size][size],
                       float cleaner_array[size], double global_array[N][M
14   // Use no global arrays
   }
```

Basic Par4All coding rules (I)

- Par4All parallelizes loop-nests made from Fortran `DO` or C99 `for` loops similar to `DO`-loops
  - Same constraints as `for`-loop accepted in OpenMP standard
  - `for ([int]` *`init-expr`*`;` *`var relational-op b`*`;` *`incr-expr`*`)` `statement`
  - Increment and bounds: integer expressions, loop-invariant
  - *`relational-op`* only <, <=, >=, >
  - Do not modify loop index inside loop body
  - Do not use `assert()` or compile with `-DNDEBUG` inside a loop. Assert has potential exit effect
  - No `goto` outside the loop, `break`, `continue`
  - No `exit()`, `longjump()`, `setcontext()`...

Basic Par4All coding rules (II)

- Data structures
  - Pointers
    - Do not use pointer arithmetics

```
1  double a[N], b[N];
   double *s = a, *d = b;
3  // Bad:
   for(int i = 0; i < N; i++)
5    *d++ = *s++;
   // Good:
7  for(int i = 0; i < N; i++)
     b[i] = a[i];
```

  - Arrays
    - PIPS uses integer polyhedron lattice in analysis, so us affine reference in parallelizable code

```
1  // Good:
2  a[2*i-3+m][3*i-j+6*n]
   // Bad (polynomial)
4  a[2*i*j][m*n-i+j]
```

Basic Par4All coding rules *(III)*

- Do not use linearized arrays

```
1  double a[n][m][l];
2  double * p = a;
   // Ugly:
4  for(int i = 0; i < n; i++)
     for(int j = 0; j < m; j++)
6      for(int k = 0; k < l; k++)
         p[m*l*i + l*j + k] = ...;
8  // Nice:
   for(int i = 0; i < n; i++)
10   for(int j = 0; j < m; j++)
       for(int k = 0; k < l; k++)
12       a[i][j][k] = ...;
```

- Do not use recursion

Parallelizing only libraries

- The code to parallelize may be used by a big framework (Matlab, Scilab...)
- ⇝ Impossible to parallelize whole application
- Limit on code to parallelize
- Par4All is interprocedural compiler, use some global information...
- May be useful to build a fake `main()` that call all library elements in a typical way so that Par4All can parallelize them
- This `main()` can do unit tests, performance measures...
- To build final application: drop fake `main()` and compile/link parallelized `.p4a` files with global application
- Concept close to the Par4All stubs

Outline

Future challenges for the real world (I)

- Make a compiler with features that compose: able to generate heterogeneous code for heterogeneous machines with all together:
  - MPI code generation between nodes
  - Generate OpenMP parallel code for SMP Processors inside node
  - Multi-GPU with each SMP thread controlling a GPU
  - Work distribution (*à la *PU*?) between GPU and OpenMP
  - Generate CUDA/OpenCL GPU or other accelerator code
  - Generate SIMD vector code in OpenMP
  - Generate SIMD vector code in GPU code
  - Generating KPN-style task code for special accelerators (SCMP...) [SCALOPES, SMECY] or cloud-computing
- Source-to-source transformation is a key technology
- Rely a lot on Par4All Accel run-time
  - Define good minimal abstractions
  - Simplify compiler infrastructure

Future challenges for the real world *(II)*

- Improve target portability
- Finding a good ratio between specific architecture features and global efficiency
- Future is to static compilation + run-time optimizations...

- PyPS as a generic source-to-source compiler
  - Started as Python abstraction of PIPS pass manager
  - Evolved in a generic source-to-source compiler infrastructure
  - Parallel evolution of Par4All & PyPS ⇝ refactoring of Par4All back to PyPS future features
  - Python in PyPS through multiple inheritance, mix-ins, dynamicity... helps a lot
  - Common refactoring of PyPS+Par4All soon
  - Light-weight (YAML...) interaction with Eclipse [OpenGPU]
  - Combine different source-to-source compilers at phase level
  - Use source core + `#pragma`/decorations + intrinsic functions [SMECY, SIMILAN]
  - Useful to have non-compiler expert writing specialized compilers

Future challenges for the real world *(III)*

Software engineering issue 

- How to have compiler parts combine seamlessly? ☹

Eclipse interfacing (I)

Motivation

- Not easy to understand why some code is not parallel or inefficient ☹
- ⇝ Need to represent internal representation to he user
- Need interaction between tools and user
- Useful also to debug the tools themselves ☺
- Reuse existing IDE

Eclipse interfacing (II)

Integration into Eclipse

- Invoke various compilation steps
- Need to attach information to source code
- Source code with hyperlinks and tooltips...
- Graph representation (data dependence, control flow...)
- Hiding/merging information

Some ideas

- Rely on CDT and PTP mode
- Improve existing OpenCL mode
- Define simple tool interaction framework by Eclipse expert
  - Simple textual communication (YAML...)
  - No need to develop inside Eclipse for tool providers
- Still to be defined... WIP in OpenGPU

Eclipse interfacing (III)

In PIPS: rely on attachments to have IR information flowing through the prettyprinter (back to Emacs PIPS...)

Towards multi-GPU with OpenCL & *PU

- Side effect of MediaGPU project...
- LaBRI is working on *PU to distribute kernel code on any node, on any accelerator (GPU, SPU, CPU) ⇝ *PU
- Recent developments: virtual OpenCL device that rely on *PU for execution
  - ⇝ Load balancing on CPU & GPU of kernels ☺
-  Only work on asynchronous and independent kernels
- Need new phases to detect independent kernels
- Need new phases to split loop nests into independent multi-GPU kernels that cannot shared memory...
- Some hard work to be efficient ☹
- WIP in MediaGPU

Outline

Stub abstractions (I)

- Real code ≡ source code + library code, intrinsic functions, system library...
- Parallelizer « understands » (at some level...) source code
- ⇝ How to parallelize source code with library code, intrinsic functions, system library... without source code?
- Example : a programmer uses a special IO function that is missed by the compiler ⇝ no results really used ⇝ dead-code elimination
- Use stubs and stub recipes
  - Provide stub functions to satisfy the analysis without having to describe them in the compiler
    For PIPS:
    - Equivalent memory effect for interprocedural abstract interpretation
    - Equivalent I/O effect
    - Set variable values in a polyhedral-friendly way when available

Stub abstractions (II)

- Provide some sequential implementation that can be well parallelized
- Provide some parallelized implementation for GPU
- Give information for Par4All Accel runtime or static estimator to know if it is worth to move data on GPU or execute on OpenMP on CPU if not already on GPU (reminiscent HPF (re)distribution...)
- Provide Makefiles/glue/... to compile everything
- ...

- Only discover function use during compilation
- Need to cope with dynamic linking...

On-going generalization through stub concept abstraction + stub broker

Stub abstractions (I)

- Real code ≡ source code + library code, intrinsic functions, system library...
- Parallelizer « understands » (at some level...) source code
- ⇝ How to parallelize source code with library code, intrinsic functions, system library... without source code?
- Example : a programmer uses a special IO function that is missed by the compiler ⇝ no results really used ⇝ dead-code elimination
- Use stubs and stub recipes
  - Provide stub functions to satisfy the analysis without having to describes them in the compiler
    For PIPS:
    - Equivalent memory effect for interprocedural abstract interpretation
    - Equivalent IO effect
    - Set variable values in a polyhedral-friendly way when available

Stub abstractions (II)

- Provide some sequential implementation that can be well parallelized
- Provide some parallelized implementation for GPU
- Give information for Par4All Accel runtime or static estimator to know if it is worth to move data on GPU or execute on OpenMP on CPU if not already on GPU (reminiscent HPF (re)distribution...)
- Provide Makefiles/glue/... to compile everything
- ...

- Only discover function use during compilation
- Need to cope with dynamic linking...

On-going generalization through stub concept abstraction + stub broker

Des ordinateurs et des humains... (I)

- Parallélisme : compliqué
- Implique développement harmonieux
  - Matériel
  - Modèles de programmation
  - Langages parallèles
  - Compilation
  - Algorithmique adaptée
  - Compatibilité et portabilité logicielles
  - Utilisateur raisonne séquentiellement sur des concepts...
  - Pas que des applications high-tech...

¡ Depuis 50 ans le matériel attend le logiciel !

Saint Cloud gatekeeper & massive virtual I/O

Conclusion (I)

- GPU (and other heterogeneous accelerators): impressive peak performances and memory bandwidth, power efficient
- Domain is maturing: any languages, libraries, applications, tools... Just choose the good one ☺
- Real codes are often not well written to be parallelized... even by human being ☹
- At least writing clean C99/Fortran/Scilab... code should be a prerequisite
- Take a positive attitude. . . Parallelization is a good opportunity for deep cleaning (refactoring, modernization. . .) ⇝ improve also the original code
- Open standards to avoid sticking to some architectures
- Need software tools and environments that will last through business plans or companies

Conclusion (II)

- Open implementations are a warranty for long time support for a technology (cf. current tendency in military and national security projects)
- p4a motto: keep things simple
- Open Source for community network effect
- Easy way to begin with parallel programming
- Source-to-source
  - Give some programming examples
  - Good start that can be reworked upon
-  Entry cost
-  Exit cost! ☹
  - Do not loose control on *your* code and *your* data !
- Minute de publicité subliminale:
  - HPC Project recrute
  - Stages...
  - Thèses...

How to translate “manycores” in French?

- A side effect of RenPar/SympA/CFSE 2008 in Fribourg...
- Visit of the castle of Gruyère with many old French texts with *moult* usage of *moult* and even *moultement*
- *Moult* jokes with Lionel Lacassagne and Joël Falcou

How to translate “manycores” in French?

- A side effect of RenPar/SympA/CFSE 2008 in Fribourg...
- Visit of the castle of Gruyère with many old French texts with *moult* usage of *moult* and even *moultement*
- *Moult* jokes with Lionel Lacassagne and Joël Falcou
- Lionel Lacassagne (IEF) recently coined « *moulticœur* » for “manycore”

How to translate “manycores” in French?

- A side effect of RenPar/SympA/CFSE 2008 in Fribourg...
- Visit of the castle of Gruyère with many old French texts with *moult* usage of *moult* and even *moultement*
- *Moult* jokes with Lionel Lacassagne and Joël Falcou
- Lionel Lacassagne (IEF) recently coined « *moulticœur* » for “manycore”
- “Workshop on Massively Multiprocessor and Multicore Computers” ⇝ « *Journées des Ordinateurs Moultement Moultiprocesseurs et Moulticœurs* »

[Foster and Riseman, 1972]Foster, C. C. and Riseman, E. M. (1972). Percolation of code to enhance parallel dispatching an execution. *IEEE Transactions on Computers*, C-21(12):1411–1415.

[Riseman and Foster, 1972]Riseman, E. M. and Foster, C. C. (1972). The inhibition of potential parallelism by conditional jumps. *IEEE Transactions on Computers*, C-21(12):1405–1411.

[Tjaden and Flynn, 1970]Tjaden, G. S. and Flynn, M. J. (1970). Detection and parallel execution of independent instructions. *IEEE Transactions on Computers*, C-19(10):889–895.

Outline