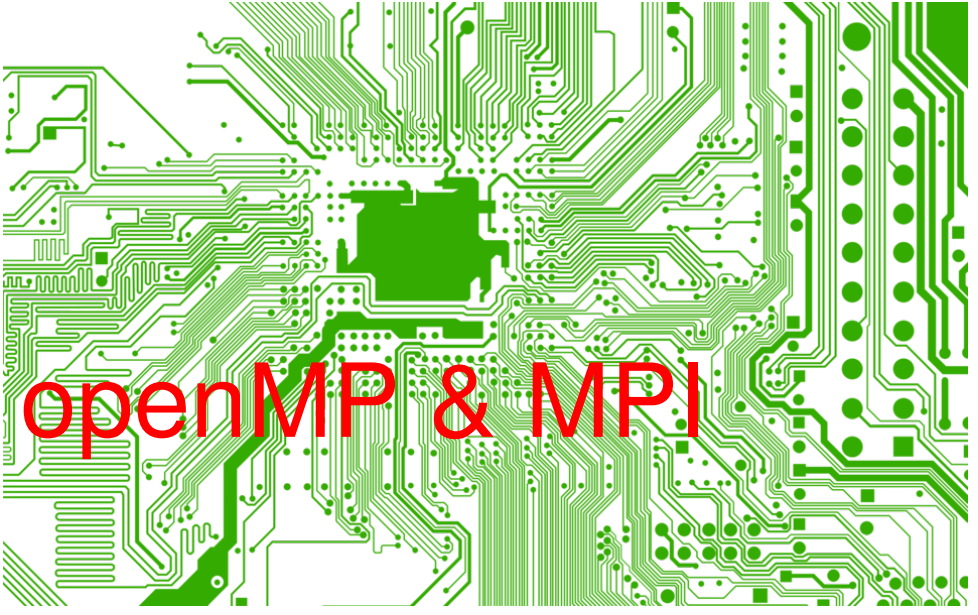
Faculté des Sciences & Techniques

Université de Limoges

Master 1ère année



openMP & MPI

Parallélisme & Application — P-F. Bonnefoi

Version du 8 novembre 2024

Table des matières

Quels sont les besoins ?









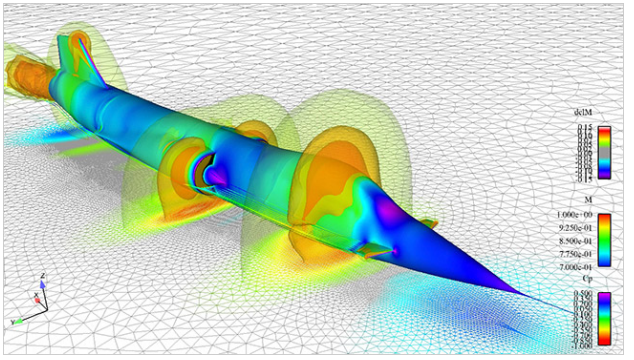


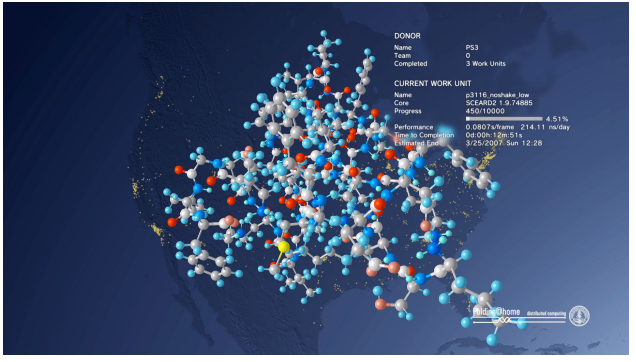

Répondre à une forte demande

En **puissance de calcul** :

- simulation, modélisation : météo, aéronautique ...
- traitement des signaux : images, sons ...
- analyse de données : génomes, fouille de données ...

Demande toujours plus importante, modèle de simulation plus complexe, obtenir des temps de calcul raisonnable, etc.



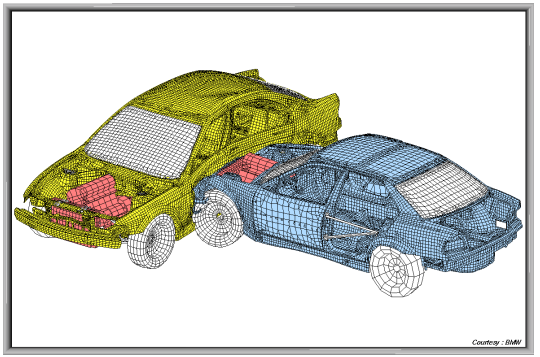


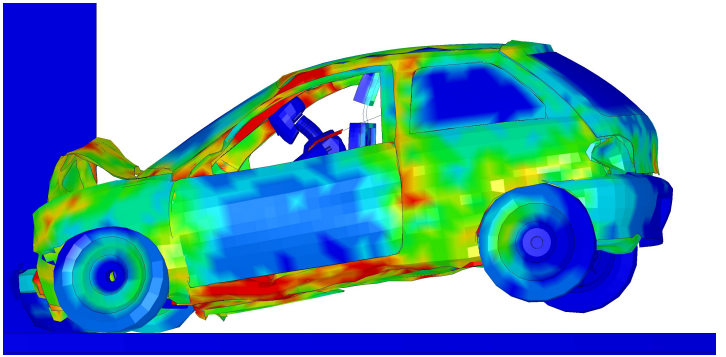
En **puissance de traitement** :

- base de données
- serveurs multimédia
- Internet

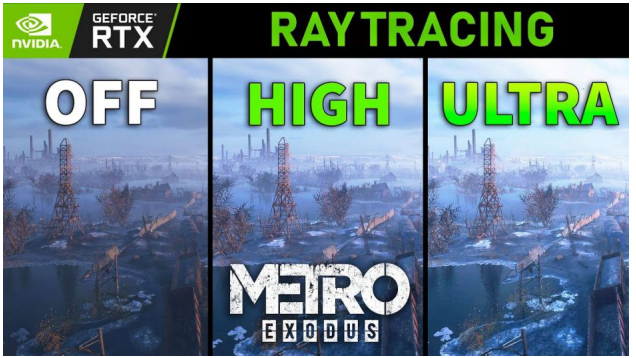
Toujours plus de données à traiter, des données plus complexes etc.

Industrie automobile

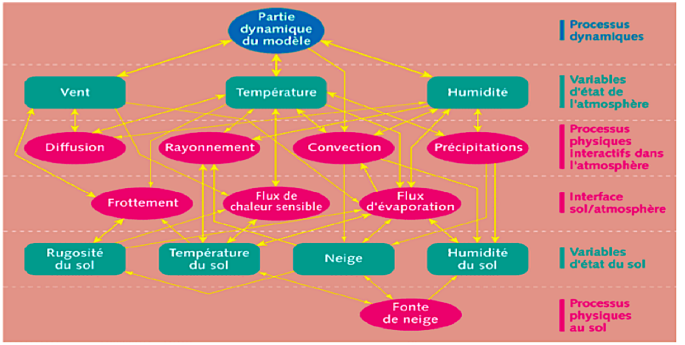




Industrie des effets spéciaux



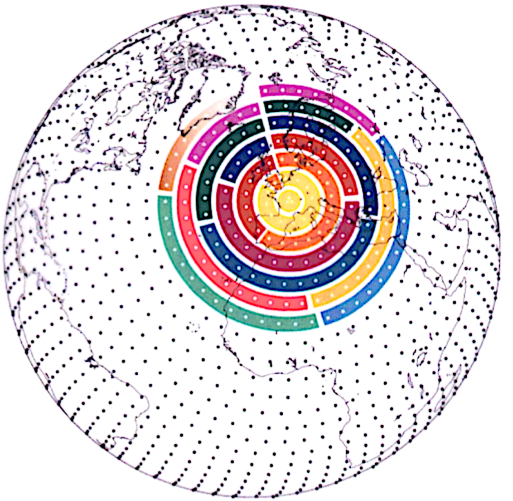
Météorologie : Modèle Arpège 1998



Découpage de l'atmosphère et répartition entre processeurs

Le nombre de variables à traiter est $Nv = 2,3.10^7$

- ▷ quatre variables à trois dimensions x 31 niveaux x 600 x 300 points sur l'horizontale ;
- ▷ une variable à deux dimensions x 600 x 300 points sur l'horizontale ;
- ▷ le nombre de calculs à effectuer pour une variable est $Nc = 7.10^3$
- ▷ le nombre de pas de temps pour réaliser une prévision à 24 heures d'échéance est $Nt = 96$ (pas de temps de 15 minutes simulées).



Puissance de calcul

Elle est exprimée en :

– **MIPS**, «*Machine Instructions Per Second*» représente le nombre d'instructions effectuées par seconde ;

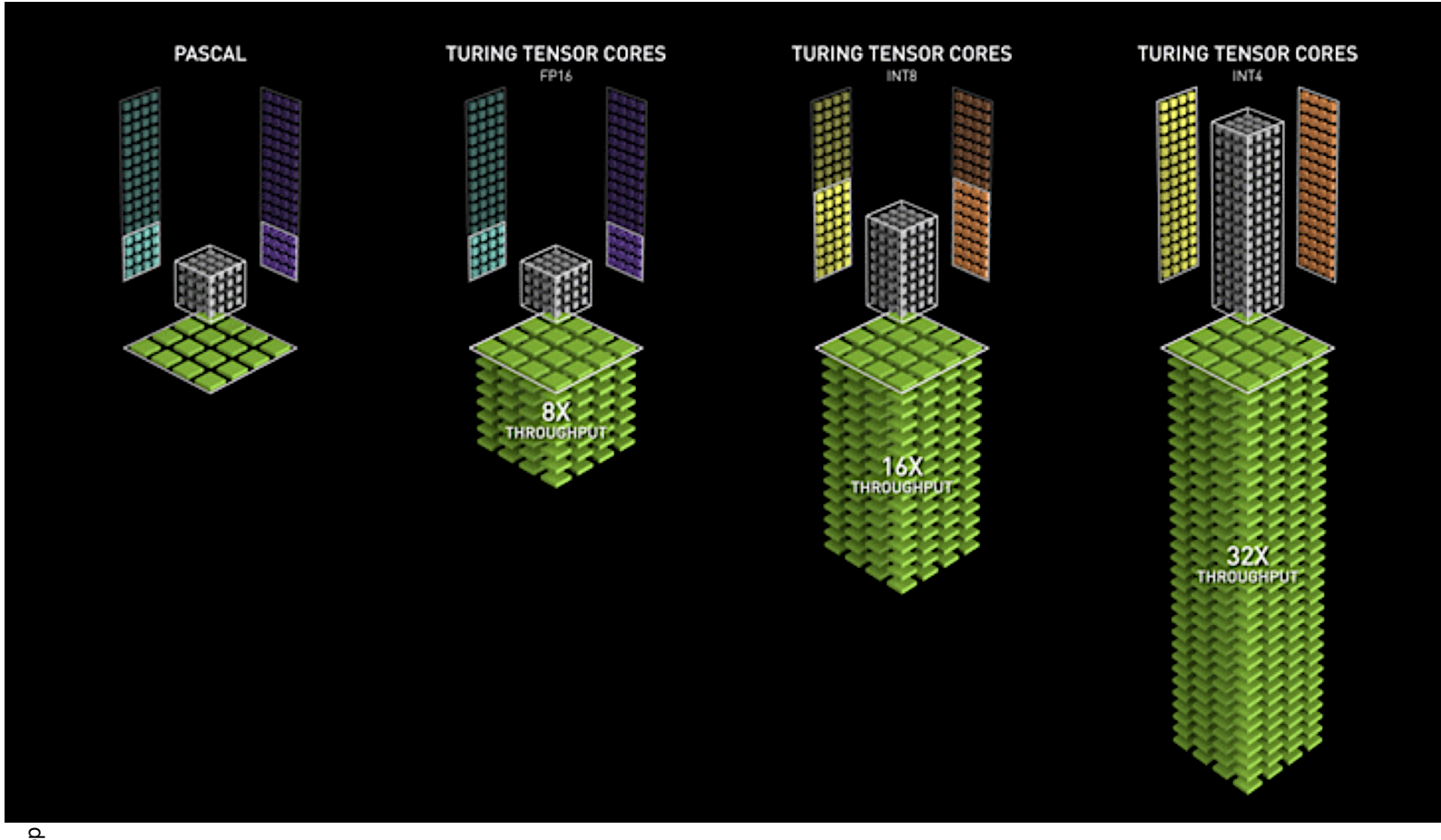
– **FLOPS** «*FLoating Point Operations Per Second*» représente le nombre d'opérations en virgule flottante effectuées par seconde ;

Les multiplicatifs : Kilo = 2^{10} ; Mega = 2^{20} ; Giga = 2^{30} ; Tera 2^{40} ; Peta 2^{50} ; Exa 2^{60}

Certains processeurs vectoriels ont une puissance de calcul de 300 Mflops par exemple.

Pour en revenir à la météorologie

- 1998 **Fujitsu VPP700** crédité d'une vitesse de calcul atteignant 62 gigaflops (62 milliards d'opérations flottantes par seconde) ;
- 2003 **Fujitsu VPP5000** avec une puissance de 1,19 Téraflops ;
- 2006 **NEC SX-8** avec une puissance de 9,1 Tflops ;
- 2021 **Sequana XH2000** développée par Bull (filiale du groupe ATOS) :
  - améliorer la prévision des **phénomènes dangereux** avec un gain de 1 à 2 heures d'échéance sur les prévisions ;
  - améliorer la précision géographique et donc mieux déterminer les risques, en descendant à une **échelle infra-départementale** ;
  - prendre en compte plus d'observations et de nouveaux types d'observations tels que les **objets connectés**.


PASCAL
TURING TENSOR CORES
FP16
8X
THROUGHPUT
TURING TENSOR CORES
INT8
16X
THROUGHPUT
TURING TENSOR CORES
INT4
32X
THROUGHPUT

Et le matériel ?

Quels sont les ordinateurs parallèles ?

1950 → 1970 : les pionniers

CDC 6600, (1964) :

- unités de calcul en parallèle,
- 10MHz,
- 2Mo,
- 3 MFlops

Utilisé par Niklaus Wirth pour définir Pascal

CDC7600 (1969) :
équivalent à 7 CDC6600 : 21 MFlops

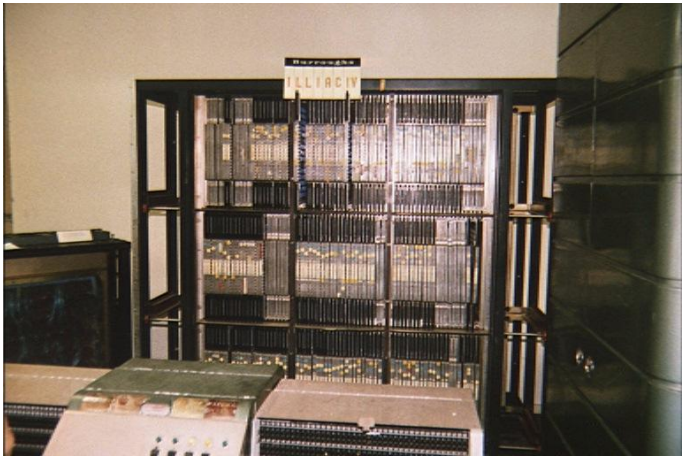
1970 → 1990 : explosion des architectures

- Cray-1 (1975), Cray X-MP (1982) : 2 à 4 processeurs, Cray-2 (1983) : 8 processeurs, Cray Y-MP (1989), Cray T3D (1993), (jusqu'à 512 processeurs, topologie : tore 3D)
- CM-5 (1992) (topologie : fat-tree).
- Hitachi S-810/820 ;
- Fujitsu VP200/VP400 ;
- Nec SX-1/2 ;
- Connection Machine 1 (65536 processeurs, topologie : hypercube) ;
- Intel iPSC/1 (128 processeurs, topologie : grille).

L'Illiac-IV 1950 à 1980

- conçu à l'Université de l'Illinois ;
- fin de construction en 1976 ;
- 64 registres de 64 bits ;
- 13MHz ;
- 1 GFlops prévu ;
- 200 MFlops obtenu ;
- Extrémement coûteux.

Des problèmes matériels : fiabilité !



Cray-1

- commercialisation ;
- utilisation du concept de pipeline ;
- 250 MFlops ;
- utilisation de micro processeurs ;
- 80 MHz.

Des problèmes de logiciel : trop difficiles !

1990 → 2000 : faillite, disparition

- fort retrait des supercalculateurs entre 1990 et 1995 ;
- **nombreuses faillites** : Thinking Machine Corporation (†), Sequent (†), Telmat (†), Archipel (†), Parsytec (†), Kendall Square Research (†), Meiko (†), BBN (†), Digital (†), IBM, Intel, CRAY (†), MasPar (), Silicon Graphics (†), Sun, Fujitsu, Nec.
- rachat de sociétés ;
- disparition des **architectures originales**.

Pourquoi ?

Manque de réalisme

- faible demande en supercalculateurs ;
- coût d'achat et d'exploitation trop élevés ;
- obsolescence rapide ;
- ratio prix/durée de vie d'une machine parallèle extrêmement élevé.

Viabilité des solutions pas toujours très étudiée

- difficultés de mise au point ;
- solutions dépassées dès leur disponibilité.

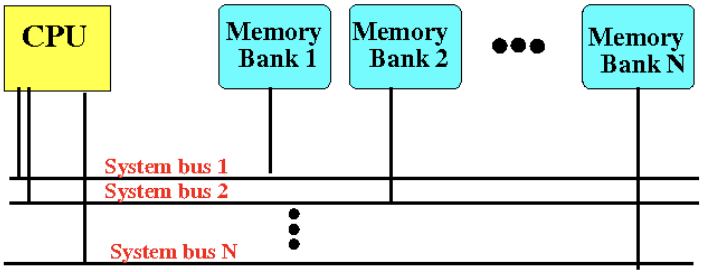
Une utilisation peu pratique

- systèmes d'exploitation propriétaires ;
- difficulté d'apprentissage.

Manque ou absence d'outils

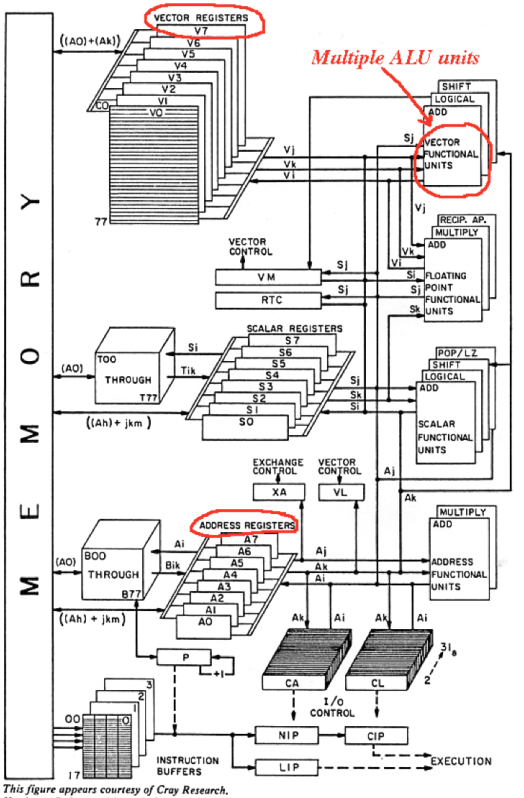
- difficulté d'exploitation.

Les prémisses des futures cartes graphiques



(Cray-1 has a 64 way interleaved memory !)

plusieurs requêtes mémoires avec différentes adresses :
jusqu'à 64 transferts simultanés s'ils sont fait sur des blocs mémoires différents !



This figure appears courtesy of Cray Research, Hardware Reference Manual, CRI publication 2240004.

Figure 10.6 CRAY-1's central processor

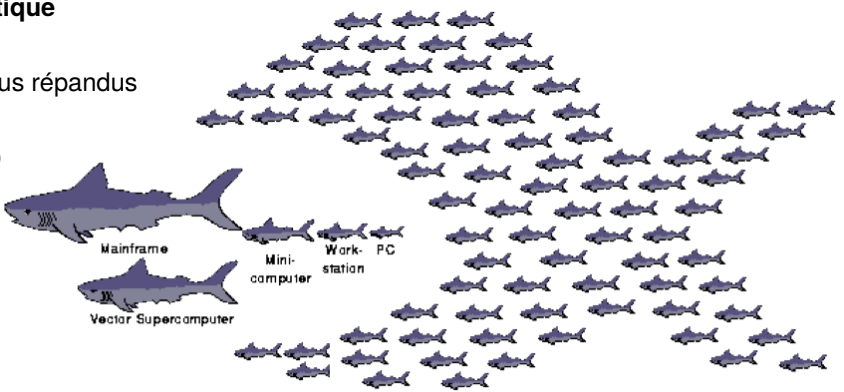
2000 : l'apparition des grilles

Améliorations apportées par la micro-informatique

– micro-processeurs rapides
– réseaux haut débit/faible latence de plus en plus répandus
– configurations PC/stations puissantes
– facilité de mise à jour (changer un composant)

Évolution du Logiciel

– bibliothèques standardisées (MPI, OpenMP)
– compilateurs parallèliseurs
– débogueurs
– système d'exploitation adapté (Beowulf)
– efforts de recherche



Disponibilité

– constat : les matériels sont la plupart du temps peu et sous-utilisés.
– Idée : utiliser ces matériels dont le nombre est énorme : meta-computing.

Grilles de calcul (metacomputing).

– Principe : des milliards de calculs indépendants effectués sur les PCs de "volontaires".
– Seti@Home : transformés de Fourier rapides,
– Folding@Home : conformation 3D de protéines.

mais...

– constat : les communications pénalisent une bonne utilisation.

Utilisation de réseaux de communication dédiés

– projet Network Of Workstation
– grappes de machines (clusters of machines)

Par ordre de puissance

`https://www.top500.org/`

Qui va arriver au PétaFlop ?

Rank	Site	Computer	Processors	Year	R_{max}	R_{peak}
1	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution IBM	212992	2007	478200	596378
2	Forschungszentrum Juelich (FZJ) Germany	JUGENE - Blue Gene/P Solution IBM	65536	2007	167300	222822
3	SGI/New Mexico Computing Applications Center (NMCAC) United States	SGI Altix ICE 8200, Xeon quad core 3.0 GHz SGI	14336	2007	126900	172032
4	Computational Research Laboratories, TATA SONS India	EKA - Cluster Platform 3000 BL460c, Xeon 53xx 3GHz, Infiniband Hewlett-Packard	14240	2007	117900	170880
5	Government Agency Sweden	Cluster Platform 3000 BL460c, Xeon 53xx 2.66GHz, Infiniband Hewlett-Packard	13728	2007	102800	146430
6	NNSA/Sandia National Laboratories United States	Red Storm - Sandia/ Cray Red Storm, Opteron 2.4 GHz dual core Cray Inc.	26569	2007	102200	127531
7	Oak Ridge National Laboratory United States	Jaguar - Cray XT4/XT3 Cray Inc.	23016	2006	101700	119350

...and the winner is :

Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz , Voltaire Infiniband / 2008 IBM	129600	1105.00	1456.70	2483.47
2	Oak Ridge National Laboratory United States	Jaguar - Cray XT5 QC 2.3 GHz / 2008 Cray Inc.	150152	1059.00	1381.40	6950.60
3	NASA/Ames Research Center/NAS United States	Pleiades - SGI Altix ICE 8200EX, Xeon QC 3.0/2.66 GHz / 2008 SGI	51200	487.01	608.83	2090.00
4	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution / 2007 IBM	212992	478.20	596.38	2329.60
5	Argonne National Laboratory United States	Blue Gene/P Solution / 2007 IBM	163840	450.30	557.06	1260.00

Power data in KW for entire system
Puissance exprimée en Tflop.

En début 2011

Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	National Supercomputing Center in Tianjin China	Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C / 2010 NUDT	186368	2566.00	4701.00	4040.00
2	DOE/SC/Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00	6950.60
3	National Supercomputing Centre in Shenzhen (NSCS) China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU / 2010 Dawning	120640	1271.00	2984.30	2580.00
4	GSIC Center, Tokyo Institute of Technology Japan	TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278	1192.00	2287.63	1398.61
5	DOE/SC/LBNL/NERSC United States	Hopper - Cray XE6 12-core 2.1 GHz / 2010 Cray Inc.	153408	1054.00	1288.63	2910.00
6	Commissariat a l'Energie Atomique (CEA) France	Tera-100 - Bull bullx super-node S6010/S6030 / 2010 Bull SA	138368	1050.00	1254.55	4590.00
7	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband / 2009 IBM	122400	1042.00	1375.78	2345.50
8	National Institute for Computational Sciences/University of Tennessee United States	Kraken XT5 - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	98928	831.70	1028.85	3090.00

En début 2012

Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu	548352	8162.00	8773.63	9898.56
2	National Supercomputing Center in Tianjin China	Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C / 2010 NUDT	186368	2566.00	4701.00	4040.00
3	DOE/SC/Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00	6950.60
4	National Supercomputing Centre in Shenzhen (NSCS) China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU / 2010 Dawning	120640	1271.00	2984.30	2580.00
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	73278	1192.00	2287.63	1398.61
6	DOE/NNSA/LANL/SNL United States	Cielo - Cray XE6 8-core 2.4 GHz / 2011 Cray Inc.	142272	1110.00	1365.81	3980.00
7	NASA/Ames Research Center/NAS United States	Pleiades - SGI Altix ICE 8200EX/8400EX, Xeon HT QC 3.0/Xeon 5570/5670 2.93 Ghz, Infiniband / 2011 SGI	111104	1088.00	1315.33	4102.00
8	DOE/SC/LBNL/NERSC United States	Hopper - Cray XE6 12-core 2.1 GHz / 2010 Cray Inc.	153408	1054.00	1288.63	2910.00
9	Commissariat a l'Energie Atomique (CEA) France	Tera-100 - Bull bullx super-node S6010/S6030 / 2010 Bull SA	138368	1050.00	1254.55	4590.00
10	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband / 2009 IBM	122400	1042.00	1375.78	2345.50

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560640	17590.0	27112.5	8209
2	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1572864	16324.8	20132.7	7890
3	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705024	10510.0	11280.4	12660
4	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786432	8162.4	10066.3	3945
5	Forschungszentrum Juelich (FZJ) Germany	**JUQUEEN** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393216	4141.2	5033.2	1970
6	Leibniz Rechenzentrum Germany	**SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR IBM	147456	2897.0	3185.1	3423
7	Texas Advanced Computing Center/Univ. of Texas United States	**Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi Dell	204900	2660.3	3959.0	
8	National Supercomputing Center in Tianjin China	**Tianhe-1A** - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 NUDT	186368	2566.0	4701.0	4040

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
6	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984	6,271.0	7,788.9	2,325
7	Texas Advanced Computing Center/Univ. of Texas United States	**Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462,462	5,168.1	8,520.1	4,510
8	Forschungszentrum Juelich (FZJ) Germany	**JUQUEEN** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	458,752	5,008.9	5,872.0	2,301










RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
6	DOE/NNSA/LANL/SNL United States	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc.	301,056	8,100.9	11,078.9	
7	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984	6,271.0	7,788.9	2,325

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
4	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890
5	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/SC/LBNL/NERSC United States	622,336	14,014.7	27,880.7	3,939

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

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,397,824	143,500.0	200,794.9	9,783
2	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
5	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	387,872	21,230.0	27,154.3	2,384

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
2	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
5	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR , Dell EMC Texas Advanced Computing Center/Univ. of Texas United States	448,448	23,516.4	38,745.9	

Processeurs ARM !

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,299,072	415,530.0	513,854.7	28,335
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482

ExaFlop !

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,730,112	1,102.00	1,685.65	21,100
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	1,110,144	151.90	214.35	2,942
4	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096
5	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94.64	125.71	7,438

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,206.00	1,714.81	22,786
2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
3	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
4	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
5	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107

Hum...

Comment s’y retrouver ?

Notions de flot de calcul et de flot de données

Sur tout type de machine, un algorithme consiste en un **flot d'instructions** à exécuter sur un **flot de données**.

On a **quatre modèles** de calcul suivant qu'il existe un ou plusieurs de ces flots :

- ▷ Modèle SISD : *Single Instruction Single Data* ;
- ▷ Modèle MISD : *Multiple Instructions Single Data* ;
- ▷ Modèle SIMD : *Single Instruction Multiple Data* ;
- ▷ Modèle MIMD : *Multiple Instruction Multiple Data*.

Classification de Flyn

Flot d'instructions \ Flot de données	Unique	Multiple
Unique	SISD (Von Neumann)	SIMD (tab de processeurs)
Multiple	MISD (pipeline)	MIMD (multiprocesseurs)

SISD

Notre ordinateur ? mais il est déjà superscalaire, multi-coeur...

MISD

Les machines vectorielles multi-processeurs :

- peut exécuter plusieurs instructions en même temps sur la même donnée (processeurs vectoriels et architectures pipelines)
- faible nombre de processeurs puissants (1 à 16)
- mémoire partagée
- limite atteinte, coût important

SIMD

Les machines **synchrones** :

- très grand nombre d 'éléments de calcul (4096 à 65536) de faible puissance avec une toute petite mémoire locale
- un programme unique : exécution d'une même instruction sur des données différentes : GPU

MIMD

Les multi-processeurs à **mémoires distribuées** :

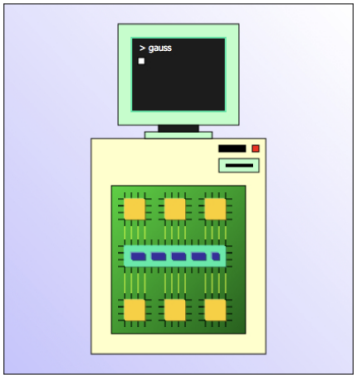
- grand nombre de processeurs ordinaires à mémoire locale
- communication par envoi de messages à travers des réseaux de communication
- chaque processeurs a son propre programme

Les multi processeurs à **mémoire partagée** :

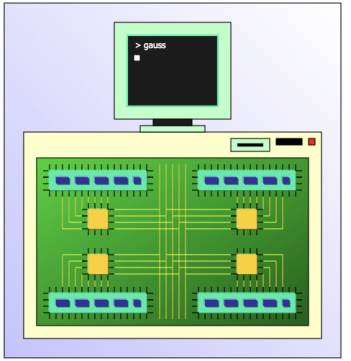
- Si le temps d'accès est égal pour chaque processeur à la mémoire, on parle de UMA, «*Uniform Memory Acess*», ou «*Symmetric Multiprocessors*» (SMP) Exemple : un Core 2 Duo ou multi-cores…
- Si le temps d'accès n'est pas le même on parle de NUMA : «*Non Uniform Memory Access*».

Différentes approches matérielles

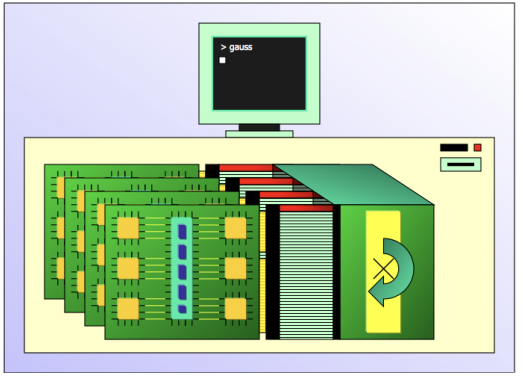
machine à **mémoire partagée**



machine à **mémoire distribuée**



machine **hybrides** : «*Non-Uniform Memory Access*»

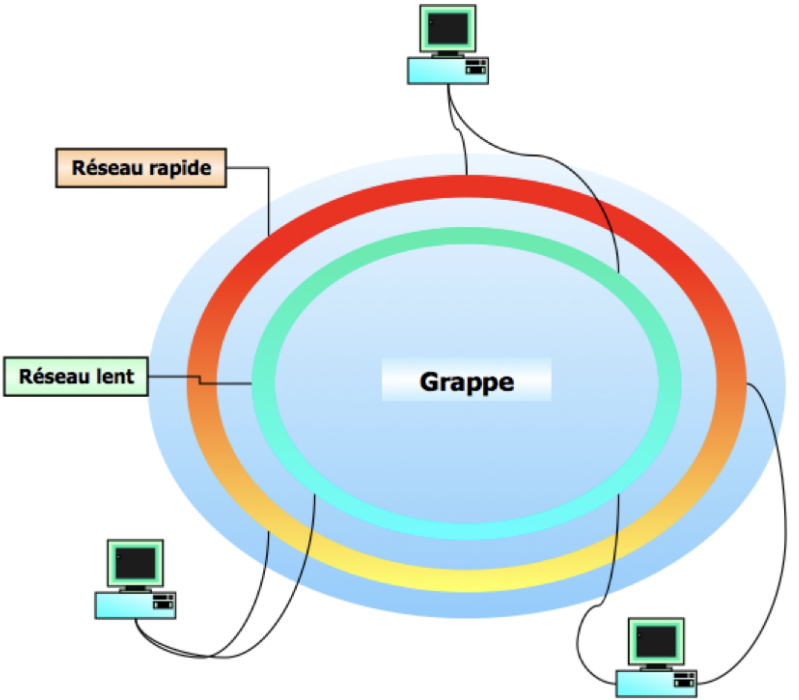


Ferme, grappe ou cluster

Un ensemble de machines
- des PC du commerce

Un réseau classique :
- lent
- réservé à l'administration

Un réseau rapide
- temps de transfert réduit
- débit élevé
- réservé aux applications



Réseau InfiniBand de la société Mellanox

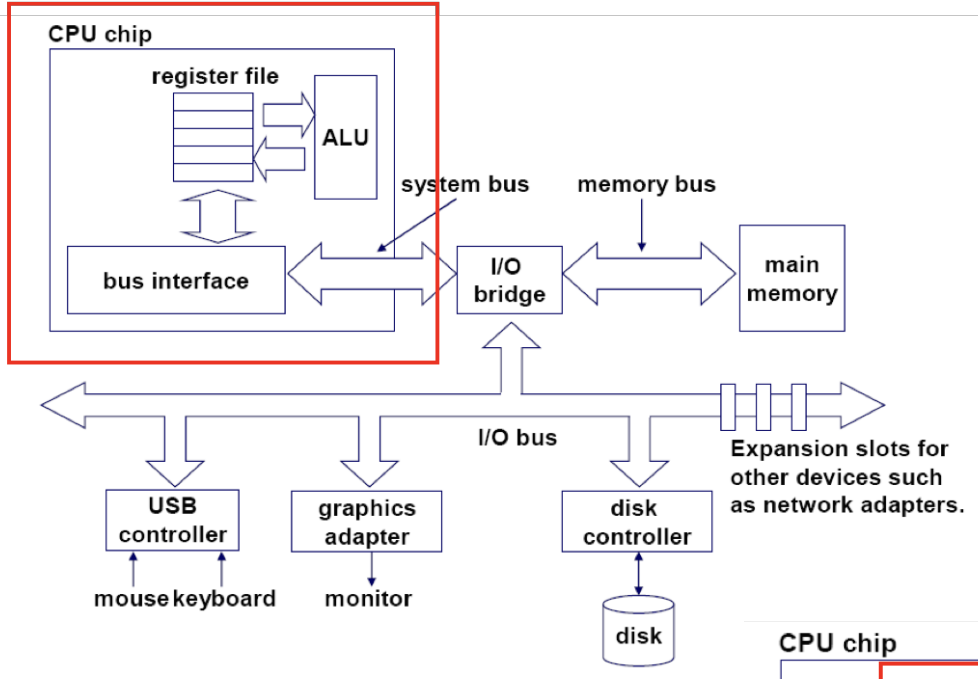
Wikipedia

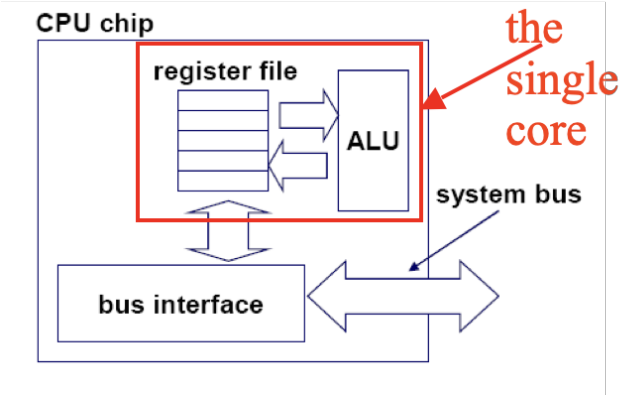
InfiniBand (IB) is a computer networking communications standard used in high-performance computing that features **very high throughput** and **very low latency**.
As of 2014, it was the most commonly used interconnect in supercomputers. In 2016, Ethernet replaced InfiniBand as the most popular system interconnect of TOP500 supercomputers.

Characteristics										
		SDR	DDR	QDR	FDR10	FDR	EDR	HDR	NDR	XDR
Signaling rate (Gbit/s)		2.5	5	10	10.3125	14.0625	25.78125	50	100	250
Theoretical effective throughput (Gb/s)	for 1 link	2	4	8	10	13.64	25	50	100	250
	for 4 links	8	16	32	40	54.54	100	200	400	1000
	for 8 links	16	32	64	80	109.08	200	400	800	2000
	for 12 links	24	48	96	120	163.64	300	600	1200	3000
Encoding (bits)		8b/10b			64b/66b				PAM4	t.b.d.
Adapter latency (µs)		5	2.5	1.3	0.7	0.7	0.5	less?	t.b.d.	t.b.d.
Year		2001 2003	2005	2007	2011	2011	2014	2018	2021	after 2023?

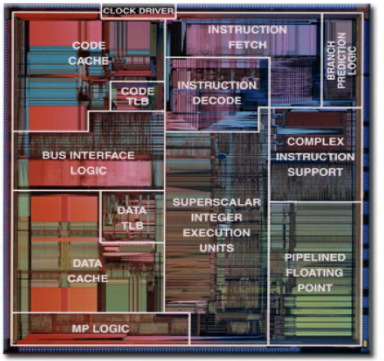
$\Longrightarrow$2019: Nvidia acquired Mellanox for $6.9B

Et les multi-cores ?

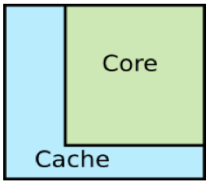




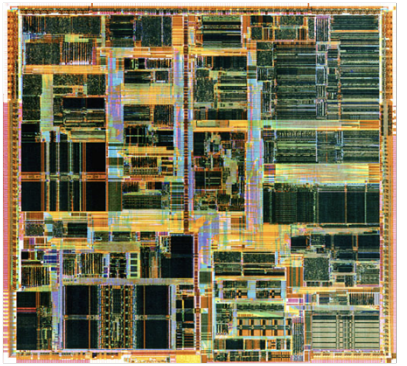
Pentium I

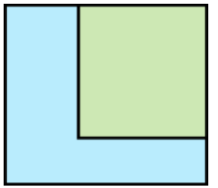


Chip area breakdown

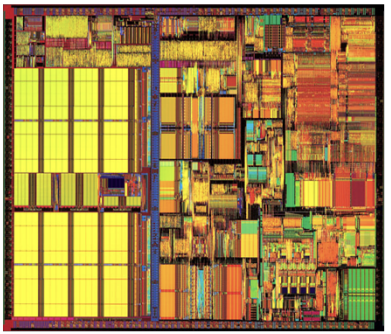


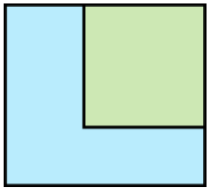
Pentium II



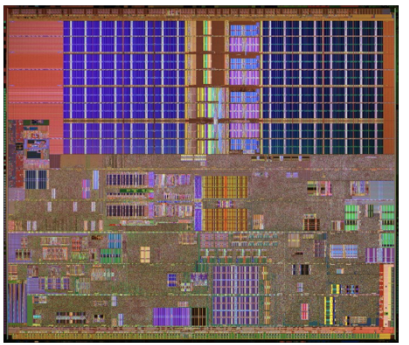


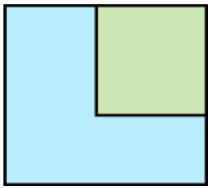
Pentium III





Pentium IV

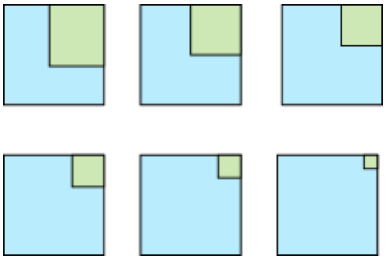




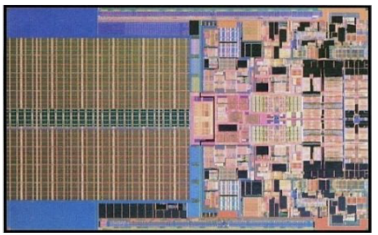
Et les multi-cores ?

L'avenir ?

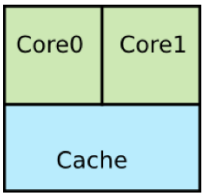
Non ! le multi-cœurs :



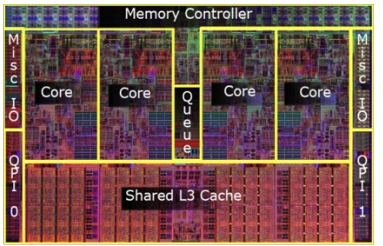
Penryn

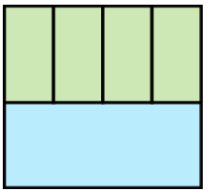


Chip area breakdown

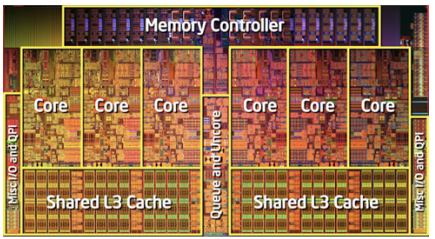


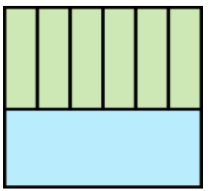
Bloomfield



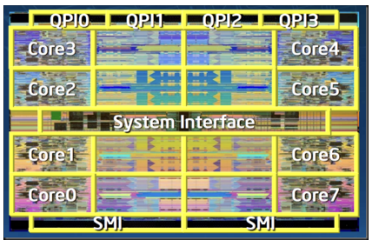


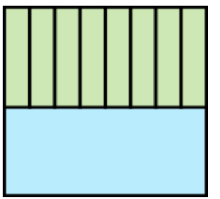
Gulftown

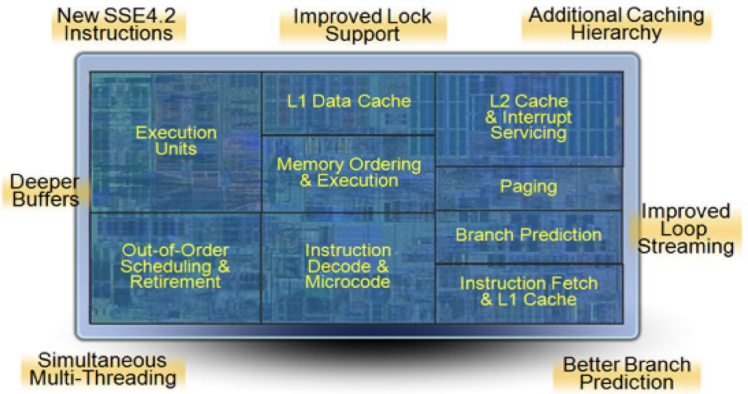


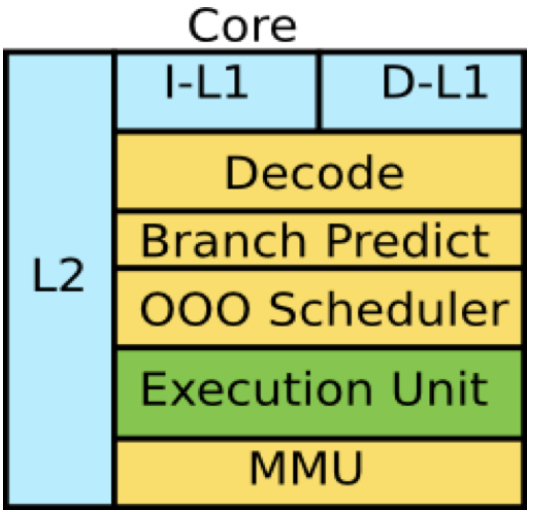


Beckton





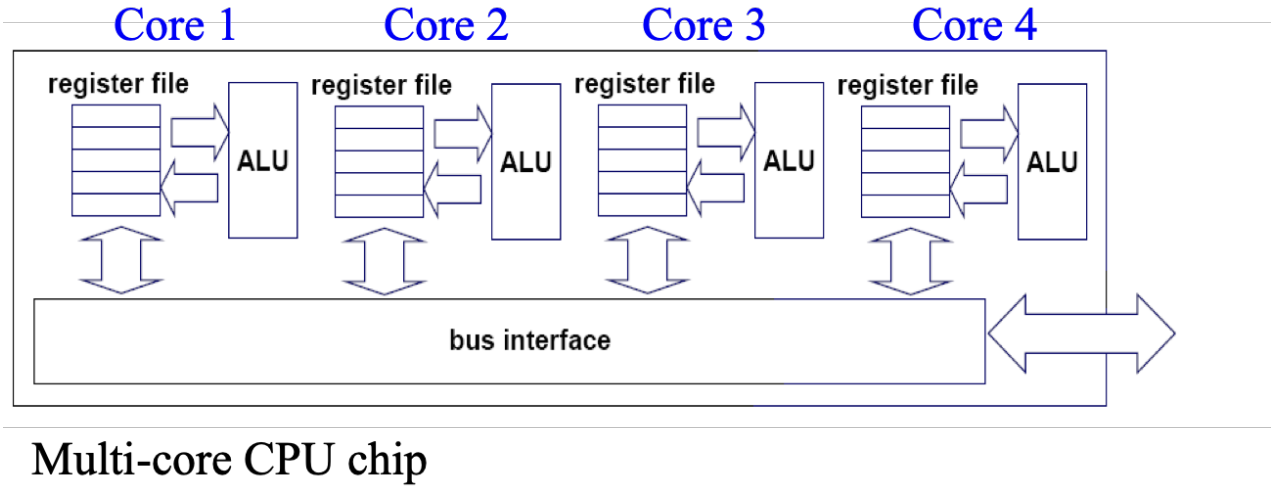




Moins de 10%de la surface sert à l'exécution rélle

OOO : «Out Of Order»

Et les «*multi-cores*» ?



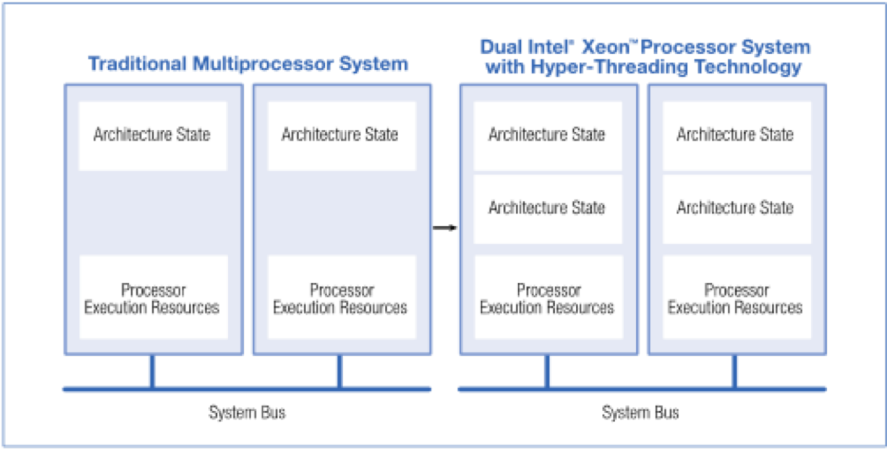
Multi-core CPU chip

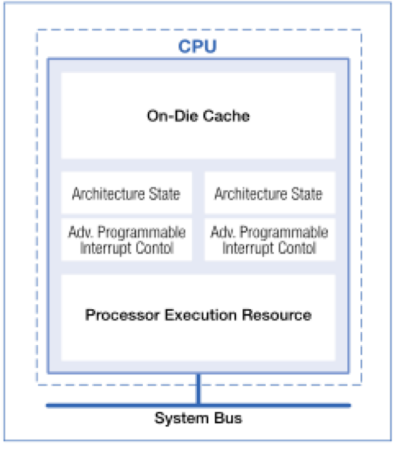
- On «*grave*» plusieurs processeurs sur le même support.
- Chaque cœur est vu par le système d'exploitation comme un **processeur séparé**.

Avantages

- On augmente moins la cadence du processeur (échauffement, consommation, difficultés de conception)
- On va vers vers plus de parallélisme (bien !)

«*Hyperthreading*» ? Qu'est-ce que c'est ?





Hyperthreading : la notion de «*processeur logique*»

Un processeur logique

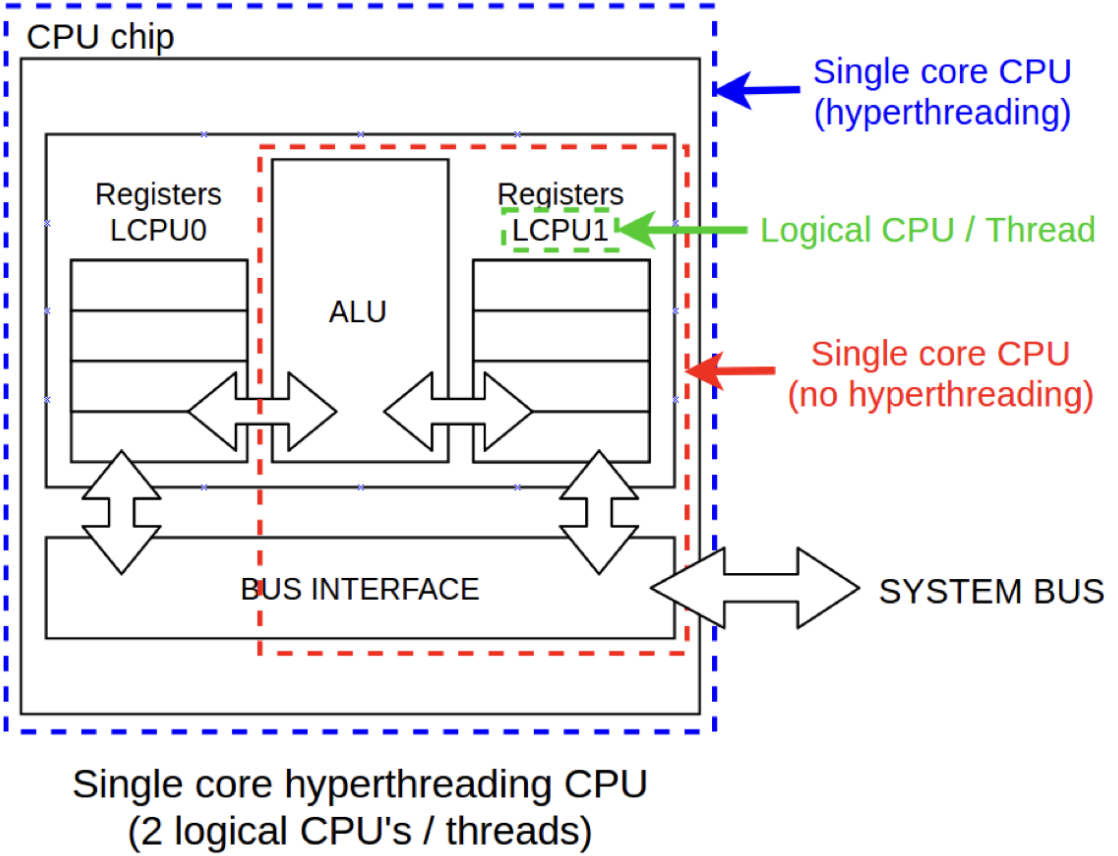
- un «*architecture state*», c-à-d un état matériel : registres, RI, CO, PSW, Interruptions ;
- son propre **flot d'instruction** ;
- peut être interrompu et stoppé **indépendamment**.

Tous les **processeurs logiques** partagent la partie «*exécution*» :

- les caches mémoires ;
- les bus mémoires ;
- le CPU, ALU, FPU, *etc*.

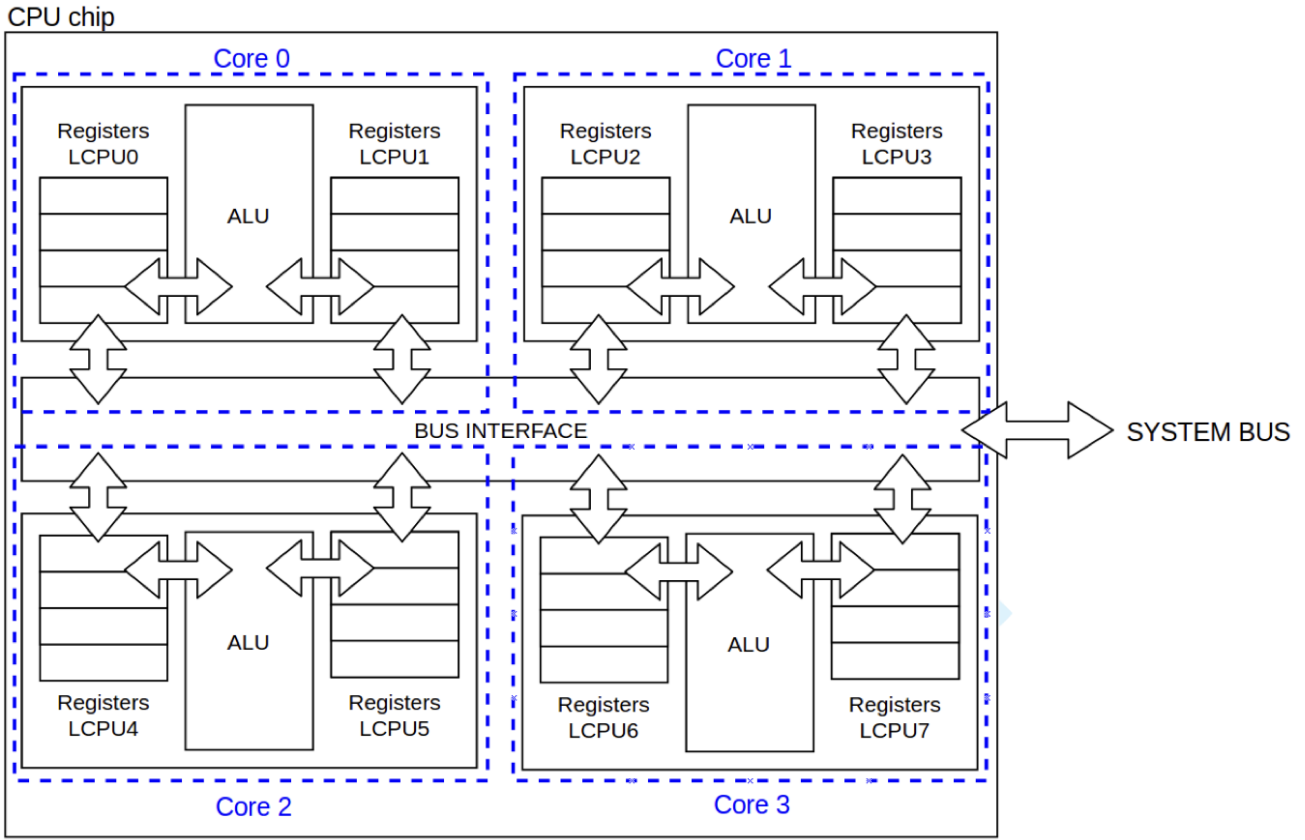
D'après la documentation d'Intel

Each logical processor maintains a complete set of the architecture state. The **architecture state** consists of registers including the general-purpose registers, the control registers, the advanced programmable interrupt controller (APIC) registers and some machine-state registers.
From a software perspective, once the architecture state is duplicated, the processor appears to be two processors. The number of transistors to store the architecture state is an extremely small fraction of the total.
Logical processors share nearly all other resources on the physical processor, such as caches, execution units, branch predictors, control logic and buses.



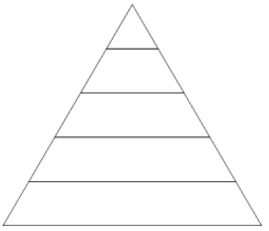
Single core hyperthreading CPU
(2 logical CPU's / threads)

Hyperthreading



Quad-core hyperthreading CPU

Hiérarchie mémoire



	Size (Byte)	Energy (pJ)	Delay (cycles)	Bandwidth (GB/s)
Reg	1K	10	1	1000
L1	32K	20	5	100
L2	256K	100	10	100
L3	8M	200	50	100
Off-chip	4G	2000	100	10

Consommation d'énergie à $45nm$:

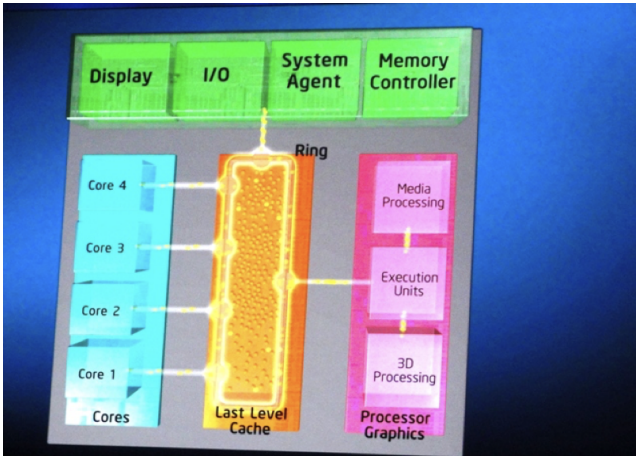
- 64bits Int ADD consomme $1pJ$;
- 64bits FP FMA consomme $200pJ$;

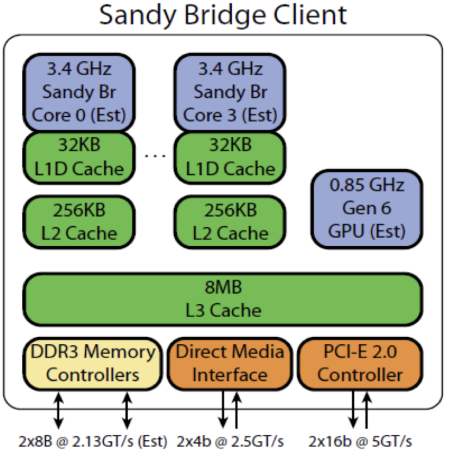
Difficile d'augementer la densité des processeurs : $7nm$ en 2023

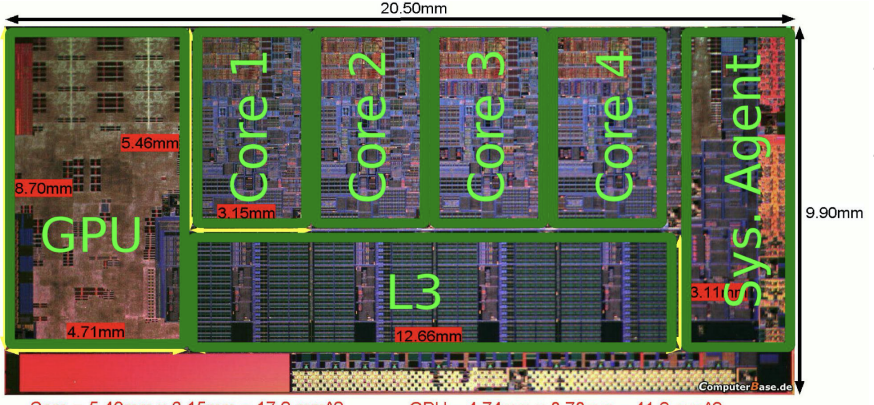
Unreleased Intel Mainstream Desktop CPU Series Specs

VideoCardz.com	Rocket Lake-S	Alder Lake-S	Raptor Lake-S	Meteor Lake-S	Lunar Lake-S
Launch Date	March 30, 2021	Q4 2021	2022	2023 (?)	2024 (?)
Fabrication Node	14nm	10nm Enhanced SuperFin	10nm Enhanced SuperFin (?)	7nm Enhanced SuperFin (?)	TBC
Core µArch	Cypress Cove	Golden Cove + Gracemont	Golden Cove + Gracemont (?)	Redwood Cove + Gracemont (?)	TBC
Graphics µArch	Gen12.1	Gen12.2	Gen12.2	Gen 12.7	Gen 13
Max Core Count	up to 8 cores	up to 16 (8+8)	up to 16 (8+8)	TBC	TBC
Socket	LGA1200	LGA1700	LGA1700	LGA1700	TBC
Memory Support	DDR4	DDR4/DDR5	DDR5	DDR5	DDR5
PCIe Gen	PCIe 4.0	PCIe 5.0	PCIe 5.0	PCIe 5.0	PCIe 5.0
Intel Core Series	11th Gen Core-S	12th Gen Core-S	13th Gen Core-S	14th Gen Core-S	14th Gen Core-S
Motherboard Chipsets	Intel 500 (Z590)	Intel 600 (eg. Z690)	TBC	TBC	TBC

Intel Sandy Bridge





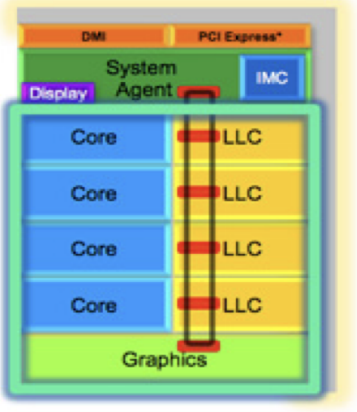


Highlight

- reconfigurable shared L3 cache for CPU and GPU
- ring bus

Sandy Bridge LLC Sharing

- **LLC shared** among all Cores, Graphics and Media
  - Graphics driver controls **which streams** are cached/coherent
  - **Any agent** can access all data in the LLC, independent of who allocated the line, after **memory range checks**
- Controlled LLC **way allocation** mechanism to prevent thrashing between Core/graphics
- Multiple coherency domains
  - **IA Domain** (*Fully coherent via cross-snoops*)
  - **Graphic domain** (*Graphics virtual caches, flushed to IA domain by graphics engine*)
  - **Non-Coherent domain** (*Display data, flushed to memory by graphics engine*)



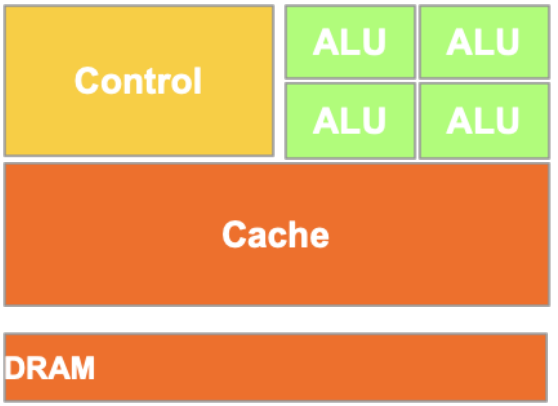
Much higher Graphics performance, DRAM power savings, more DRAM BW available for Cores

IDF2010

d'après l'article "Intel's Sandy Bridge Architecture Exposed", from Anandtech.

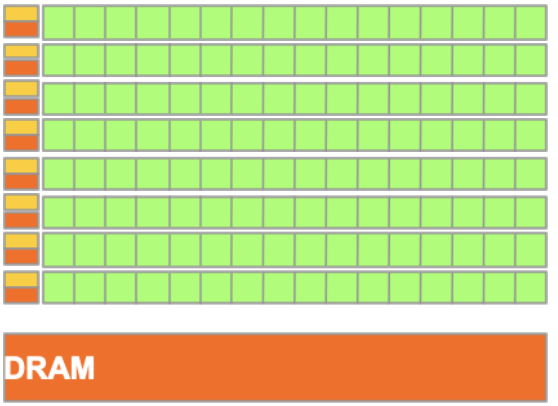
Et les GPUs ?

Prendre de la place pour des cœurs et du cache… Et si on prenait toute la place pour des CPUs ?



CPU

- ▷ Accès mémoire irréguliers ;
- ▷ Plus de cache et contrôle ;
- ▷ Cherche la **performance** par thread.

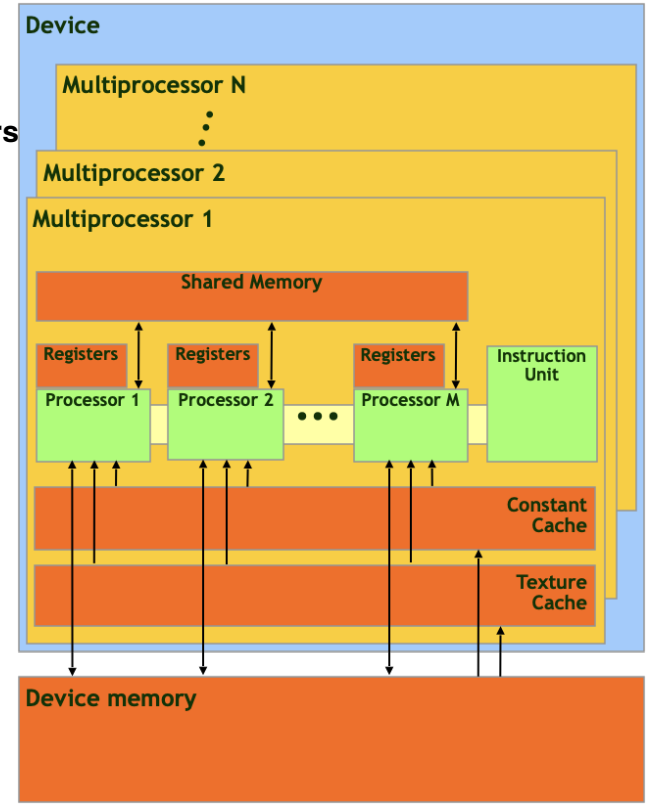


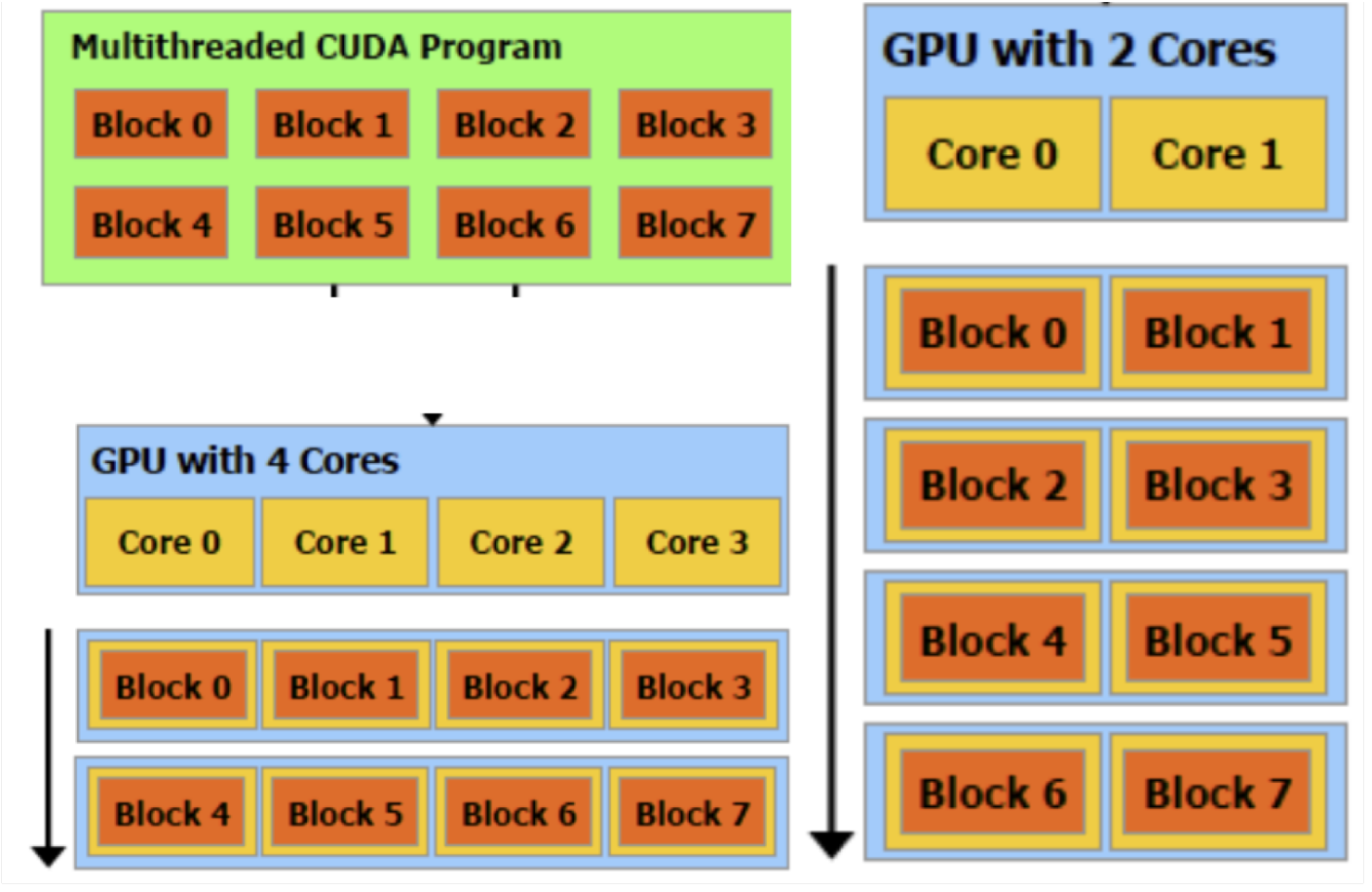
GPU

- ▷ Accès mémoire réguliers ;
- ▷ Plus d'ALUs et massivement parallèle ;
- ▷ Maximiser le **débit**.

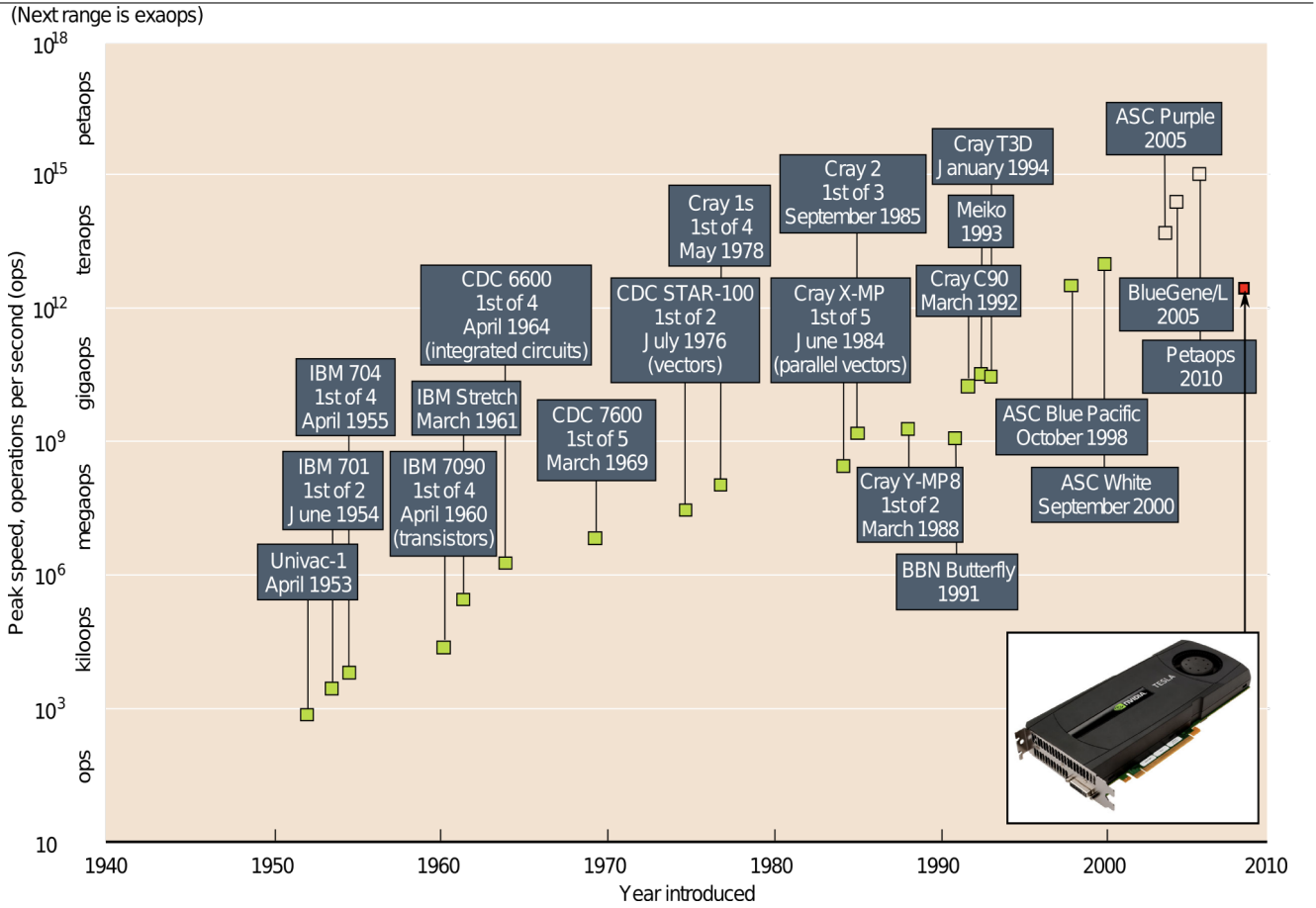
Une architecture complexe

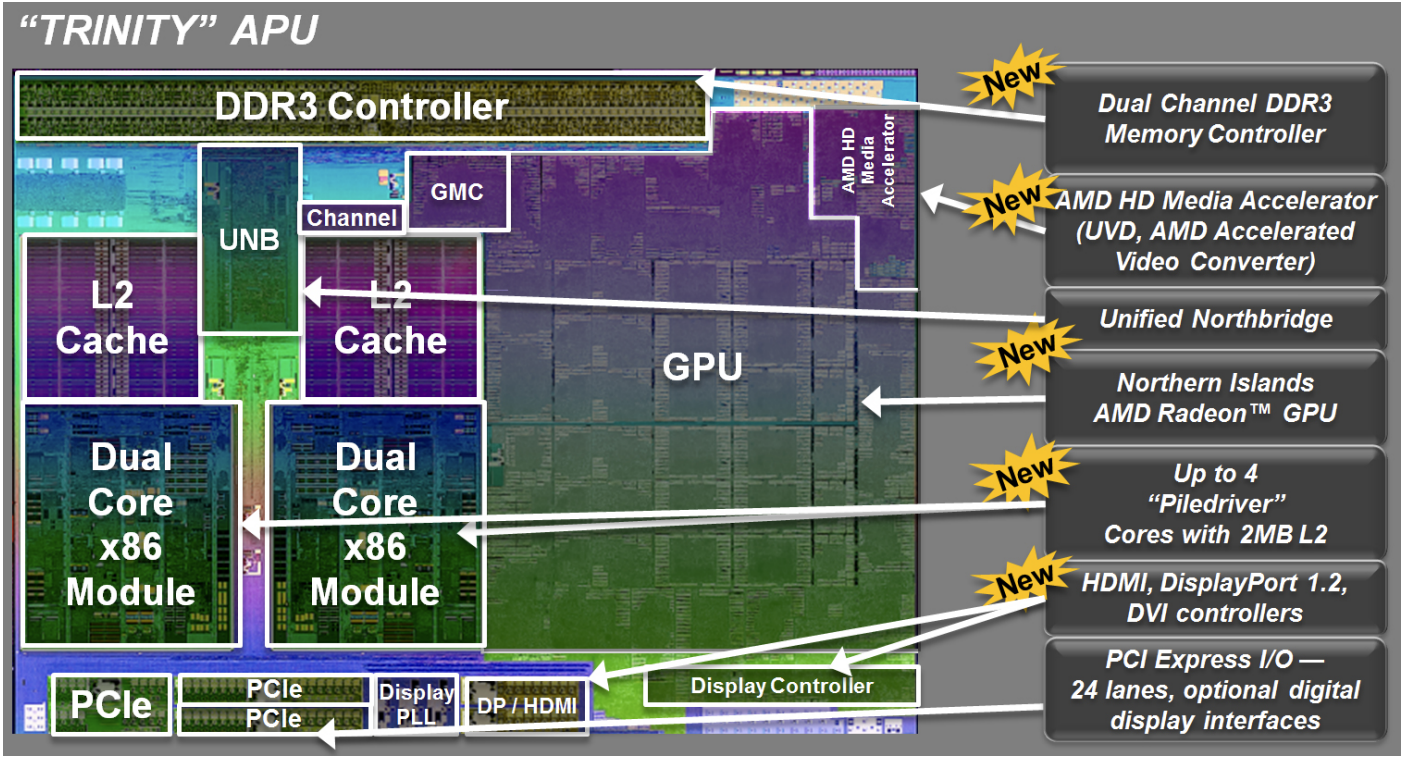
- CUDA, «*Compute Unified Device Architecture*» ;
- **Architecture hiérarchique** ;
  - Une carte contient **plusieurs multiprocesseurs**
  - Plusieurs «*cuda cores*» par multiprocesseur (32 en général)
  - Une unité de contrôle unique.
- **Différents espaces mémoires**
  - Mémoire de la carte : GDDR
    - Beaucoup de mémoire avec un bus rapide vers le multiprocesseur
  - Registres sur la puce : environ 16k
  - Mémoire partagée sur la puce :
    - Partagée entre les différents cores
    - Faible latence et organisée en bloc
  - Mémoire constante (en accès lecture uniquement) et de texture ;
    - En lecture seule et avec du cache.

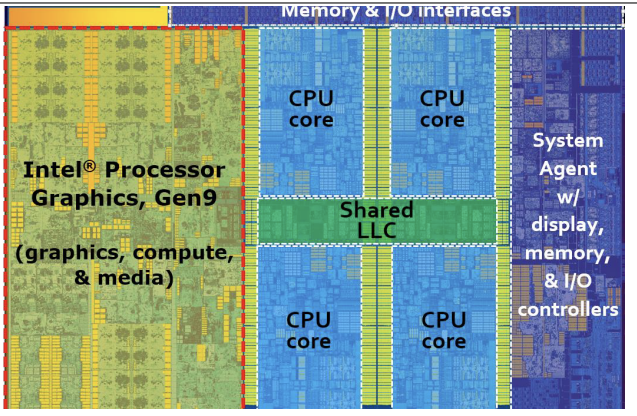


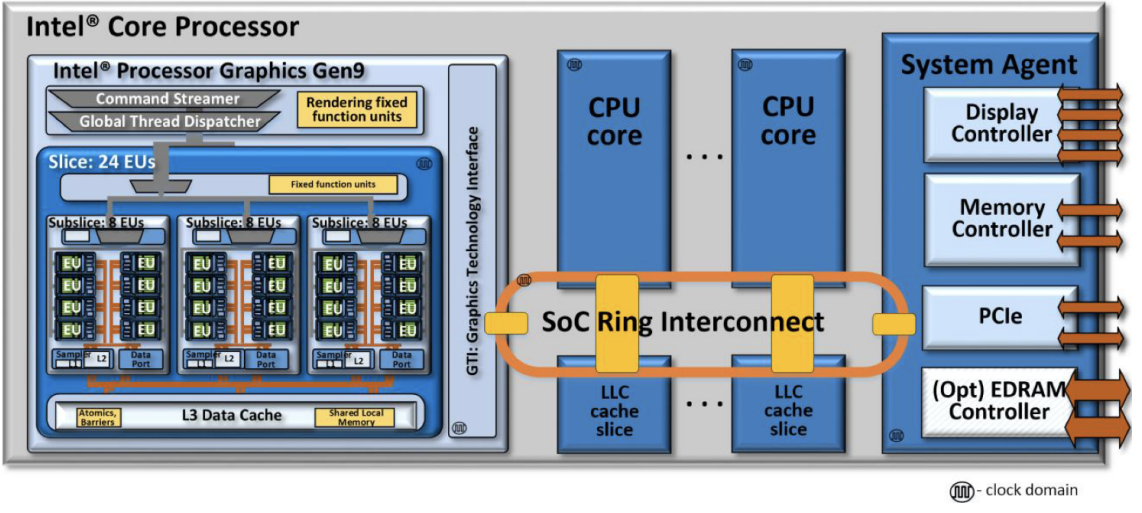

Multithreaded CUDA Program
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
GPU with 4 Cores
Core 0
Core 1
Core 2
Core 3
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
GPU with 2 Cores
Core 0
Core 1
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7

Et les GPUs ? des Résultats !




"TRINITY" APU
DDR3 Controller
UNB
Channel
GMC
L2 Cache
L2 Cache
GPU
AMD HD Media Accelerator
Dual Core x86 Module
Dual Core x86 Module
PCIe
PCIe
PCIe
Display PLL
DP / HDMI
Display Controller
New
Dual Channel DDR3 Memory Controller
New
AMD HD Media Accelerator (UVD, AMD Accelerated Video Converter)
Unified Northbridge
New
Northern Islands AMD Radeon™ GPU
New
Up to 4 "Piledriver" Cores with 2MB L2
New
HDMI, DisplayPort 1.2, DVI controllers
PCI Express I/O — 24 lanes, optional digital display interfaces


Memory & I/O Interfaces
Intel® Processor Graphics, Gen9
(graphics, compute, & media)
CPU core
CPU core
Shared LLC
CPU core
CPU core
System Agent w/ display, memory, & I/O controllers


Intel® Core Processor
Intel® Processor Graphics Gen9
Command Streamer
Global Thread Dispatcher
Rendering fixed function units
Slice: 24 EUs
Fixed function units
Subslice: 8 EUs
Subslice: 8 EUs
Subslice: 8 EUs
EU
Sampler
L2
Data Port
Atomics, Barriers
L3 Data Cache
Shared Local Memory
GTI: Graphics Technology Interface
CPU core
CPU core
SoC Ring Interconnect
LLC cache slice
LLC cache slice
System Agent
Display Controller
Memory Controller
PCIe
(Opt) EDRAM Controller
- clock domain


PCI Express 3.0 Host Interface
GigaThread Engine
GPC
L2 Cache
High-Speed Hub
NVLink

Et du point de vue du logiciel ?







L'image de la course de voiture

Plusieurs véhicules veulent aller d'un point A à un point B le plus vite possible, ils peuvent :

- ▷ faire la course sur la route et finir par :
  - ◇ soit se **suivre** les uns les autres ;
  - ◇ soit essayer de se **voler** mutuellement leurs positions respectives ;
  - ◇ soit avoir un **accident** !
- ▷ rouler sur différentes voies parallèles et arrivés ensemble s**ans entrer en collision** ;
- ▷ emprunter des **routes différentes** pour aller de A à B.

Et le parallélisme ?

- ▷ Plusieurs tâches à réaliser : chaque voiture à acheminer ;
- ▷ Chacune de ses tâches peut s'exécuter :
  - ◇ une à la fois sur un **seul processeur** : une **seule route** ;
  - ◇ en paralléle sur **plusieurs processeurs** : **plusieurs voies** sur la même route ;
  - ◇ de manière **distribuées** sur plusieurs processeurs : des **routes séparées**.
- ▷ Ces tâches nécessitent souvent d'être **synchronisées** pour éviter les collisions ou de **s'arrêter** à des feux de trafic ou bien à des panneaux de signalisation (Stop).

On peut imaginer que :

- ☐ les voitures sont des **processus** ou **threads** ;
- ☐ les routes qu'elles veulent emprunter sont des **applications** ;
- ☐ la carte des routes correspond au **matériel** ;
- ☐ et le code de la route, aux **communications** et aux **synchronisations**.

Objectif

L'objectif du parallélisme est de :

- ◇ obtenir de **meilleures performances** par rapport aux calculateurs séquentiels et vectoriels (effet pipeline essentiellement).
- ◇ traiter plus vite des **problèmes plus gros** (les machines à mémoire distribuées permettent de traiter des problèmes plus gros).

De manière informelle, une **machine parallèle** est composée de :

- ⋆ un ensemble **d'unités de calcul** (processeur) ;
- ⋆ une **mémoire** (unité de stockage) disponible :
  - ◇ soit de manière **partagée** ;
  - ◇ soit de manière **distribuée**.

Méthodologie

Un **problème original** devra être **découpé** en un certain nombre de **sous problèmes indépendants** :

- ▷ résolus **simultanément** (en parallèle)
- ▷ dont les solutions **seront combinées** pour avoir la **solution du problème original**.

Remarques

- □ La méthode est proche de celle «*diviser pour résoudre*» ⟹ algorithme **récursif**, entités indépendantes.
- □ La combinaison pose le **problème des échanges** entre unités de calcul.

Définition

Un **programme concurrent** peut contenir deux ou plus processus qui travaillent ensemble pour réaliser une tâche. *Chaque processus est un programme séquentiel ou séquence d'instructions qui sont exécutées les unes après les autres.*

- ▷ Un «*programme séquentiel*» correspond à un **seul fil de contrôle** : «*one thread of control*» ;
- ▷ Un «*programme concurrent*» possèdent **plusieurs fils de contrôle** : «*multi-threaded*» ;

Thread ou processus ?

Un processus est un programme s'exécutant au niveau d'un OS.
Une **thread** est un programme s'exécutant dans un autre programme qui est considéré comme un processus pour l'OS qui l'exécute : on parle de processus de poids léger «*lightweight processus*».

Comment travailler ensemble ?

Les processus dans un programme concurrent travaillent ensemble en communiquant les uns avec les autres.
Ces communications sont réalisées par :

- ▷ **variables partagées** : un processus écrit dans une variable qui est lue par un autre ;
- ▷ **échange de message** : un processus envoi un message qui est reçu par un autre.

Comment communiquer ?

Quelque soit la forme de communication choisie, les processus ont **besoin de se synchroniser** les uns avec les autres.

Comment se synchroniser ?

- **exclusion mutuelle** : c'est le problème de **garantir que des instructions en section critique** ne peuvent s'exécuter simultanément ;
- **synchronisation conditionnelle** : c'est le problème de **retarder un processus** jusqu'à ce qu'une condition soit vraie.

Exemple

Modèle du «*Producteur/Consommateur*» qui communiquent au travers d'une variable partagée (buffer partagé) :

- ▷ le **producteur** écrit dans le buffer ;
- ▷ le **consommateur** lit depuis le buffer.

L'exclusion mutuelle est nécessaire pour assurer que le producteur et le consommateur **n'accède pas en même temps**, permettant par exemple qu'un message écrit partiellement soit lu prématurément.

La synchronisation conditionnelle est utilisée pour **garantir qu'un message n'est pas lu** par le consommateur avant qu'il ne soit entièrement écrit par le producteur.

Vers 1960...

L'histoire de la **programmation concurrente** est liée à celle des ordinateurs.

Son émergence est liée à celle des **OS** et à l'invention des **contrôleurs de périphériques** («*device controllers*») :
– ils fonctionnent **indépendamment** du processeur central ;
– ils permettent d'effectuer des opérations d'E/S en **concurrence** d'un programme exécuté par le **processeur central**.
*Le contrôleur communique avec le processeur central par l'intermédiaire d'une **interruption**, un signal matériel qui le déroute de l'exécution de la séquence d'instructions courante pour exécuter une séquence d'instructions différente.*

Problème ?

l'intégration de contrôleur de périphérique pose le problème que certaines parties d'un programme peuvent s'exécuter dans un **ordre imprévisible** !

Ainsi, si un programme est en train de **modifier la valeur d'une variable**, une **interruption** peut arriver et peut conduire à ce qu'une autre partie du programme essaie de changer la valeur de cette **même variable** (**notion de section critique**).

Les machines multi processeurs

Il a été très vite possible de construire des machines possédant **plusieurs processeurs**.

Ces machines permettent d'exécuter un seul programme plus rapidement à condition de le réécrire pour utiliser plusieurs processeurs à la fois....

Mais :
▷ comment **synchroniser l'activité** de ces différents processeurs ?
▷ comment **utiliser plusieurs processeurs** pour accélérer un programme ?

Alors ?






Plusieurs niveaux de parallèlisme

machines

$\Longrightarrow$architectures distribuées

processeurs

$\Longrightarrow$ machines multi-processeurs

unités de calcul

$\Longrightarrow$processeurs superscalaires

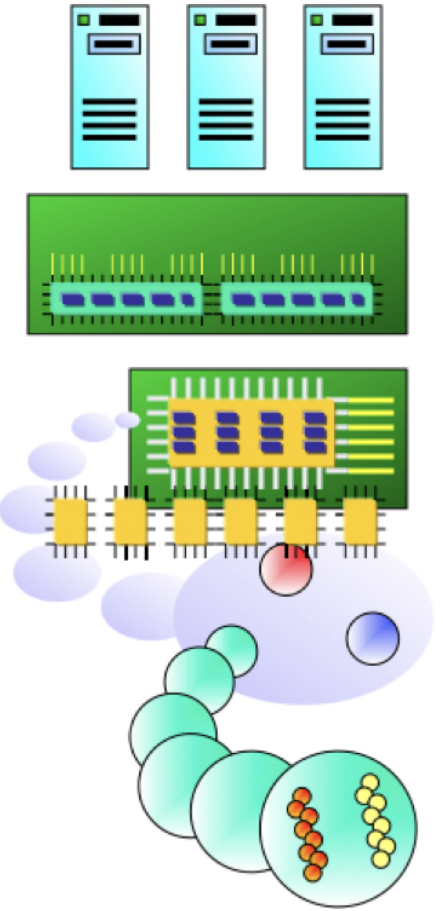
processus

$\Longrightarrow$multi-programmation

$\Longrightarrow$temps partagé

threads ou processus de poids léger

$\Longrightarrow$ multi-programmation à **grain fin**



Et l’exploitation du parallélisme ?






Accélération ou speedup

Accélération = gain de temps obtenu lors de la parallélisation du programme séquentiel.

Définition :

- ▷ Soit T_1 le temps nécessaire à un programme pour résoudre le problème A sur un ordinateur séquentiel ;
- ▷ Soit T_p le temps nécessaire à un programme pour résoudre le même problème A sur un ordinateur parallèle contenant p processeurs ;
- ▷ Alors l'accélération «*Speed-Up*» est le rapport : $S(p) = T_1 / T_p$

Cette définition n'est pas très précise

Pour obtenir des résultats comparables il faut utiliser les mêmes définitions d'**Ordinateur Séquentiel** et de **Programme Séquentiel**.

Ordinateur Séquentiel :

- □ Ordinateur // configuré avec un seul processeur ;
- □ Ordinateur séquentiel d 'une puissance similaire à l 'ordinateur //.

Programme Séquentiel :

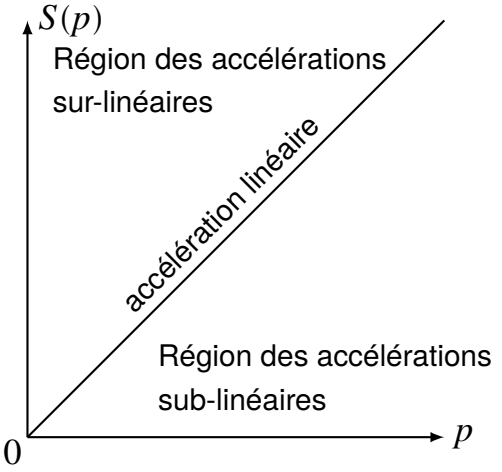
- □ Programme // configuré pour s'exécuter sur un seul processeur ;
- □ Programme séquentiel utilisant le même algo que le programme // ;
- □ Programme séquentiel le plus rapide connu utilisant le même algo que le programme // ;
- □ Programme séquentiel (ou le plus rapide) résolvant le même pb.

Beaucoup de combinaisons, il faut préciser dans chaque cas.






Accélération



Efficacité

- Soit $T_1(n)$ le temps nécessaire à l'algorithme pour résoudre une instance de problème de taille n avec un seul processeur,
- Soit $T_p(n)$ celui que la résolution prend avec p processeurs
- Soit $S(n,p) = T_1(n)/T_p(n)$ le facteur d'accélération.

On appelle **efficacité** de l 'algorithme le nombre

$$E(n,p) = S(n,p)/p$$

Efficacité = normalisation du facteur d 'accélération

Exemple

Multiplication de matrices : algorithme A moins bon que algorithme B

Algorithme A

- Temps en séquentiel : 10 minutes
- Nombre de processeurs : 10
- Temps en // : 2 minutes
- **Accélération** : $10/2 = 5$ (l'application va 5 fois plus vite)
- **Efficacité** : $5/10 = 1/2 = 0,5$

Algorithme B

- Temps en séquentiel : 10 minutes
- Nombre de processeurs : 3
- Temps en // : 4 minutes
- **Accélération** : $10/4 = 5/2 = 2,5 < 5$
- **Efficacité** : $(5/2)/3 = 0,8 > 0,5$

Loi d'Amdahl

Le temps d 'exécution T_1 d 'un programme séquentiel peut être décomposé en deux temps :

- T_s consacré à l 'exécution de la partie intrinsèquement séquentielle
- $T_{//}$ consacré à l 'exécution de la partie parallélisable

$$T_1 = T_s + T_{//}$$

Seul $T_{//}$ peut être diminué par la parallélisation.

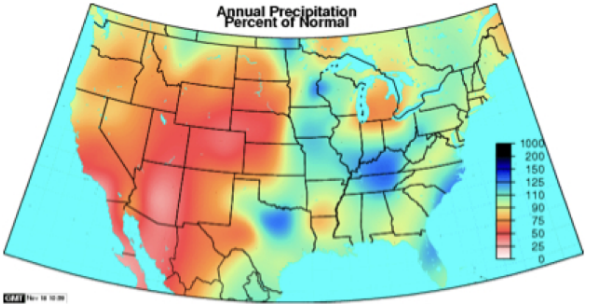
Dans le **cas idéal**, on obtiendra **au mieux** un temps $T_{//}/p$ pour la partie parallélisée.

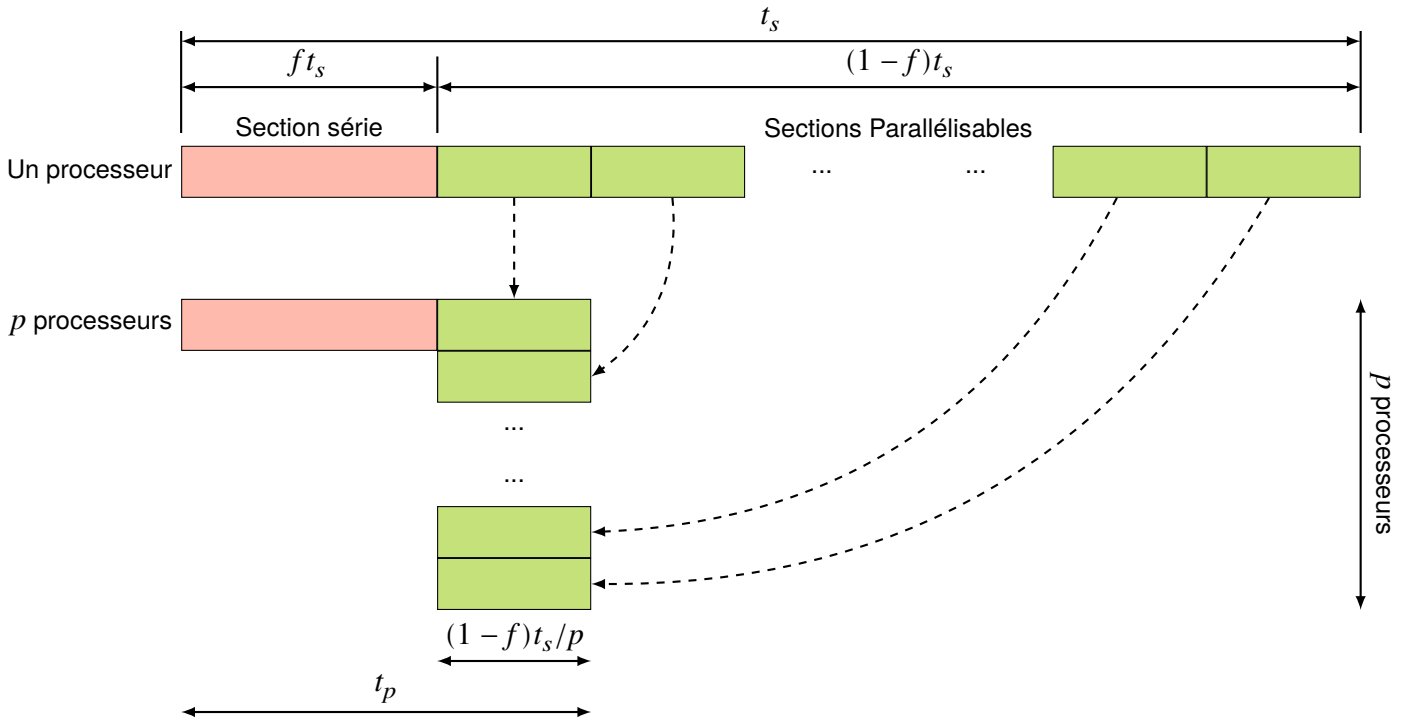
$$T_p \geq T_s + T_{//}/p$$

$\Longrightarrow$**L'accélération** d'un programme est **limitée par le pourcentage de code intrinsèquement séquentiel** qu'il contient.

Exemple : filtre graphique parallèle

- partie intrinséquement séquentielle
  - ◇ capture
  - ◇ chargement sur le serveur
- partie parallélisable
  - ◇ découpage
  - ◇ calculs pour le traitement de l'image ...





La fraction f exprime le rapport entre la partie séquentielle et parallèle par rapport au temps complet t_s :

- ft_s pour le temps de la partie séquentielle ;
- $(1-f)t_s$ pour le temps de la partie parallèle.

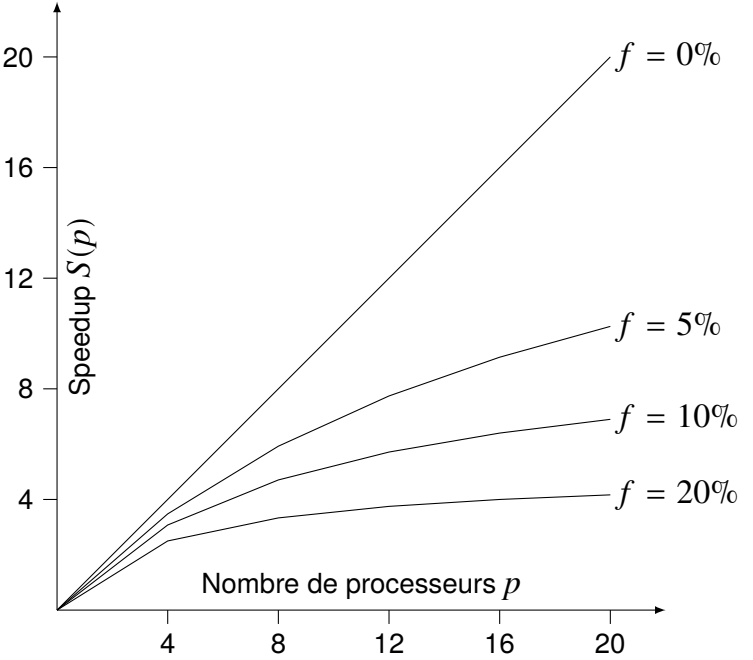
Speedup

Même avec un nombre infini de processeurs l'accélération maximale est de $1/f$

Exemple : avec seulement 5%de calcul séquentiel, le speedup maximal est de 20.

Calcul du speedup avec f:

$$S(p) = \frac{t_s}{f t_s + (1-f) t_s / p} = \frac{p}{1+(p-1)f}$$



f exprime le pourcentage de la partie séquentielle.

Une **accélération linéaire** correspond à un gain de temps égal au nombre de processeurs (100%activité)

Une **accélération sub-linéaire** implique un taux d'activité des processeurs $<$ 100 %(communication, coût du parallélisme...)

Une **accélération sur-linéaire** implique un taux d'utilisation des processeurs $>$ à 100 %ce qui paraît impossible (en accord avec la loi d 'Amdhal).

Cela se produit parfois (architecture, mémoire cache mieux adaptée que les machines mono-processeurs, utilisation de pipeline…)

Comment Paralléliser un algorithme ?





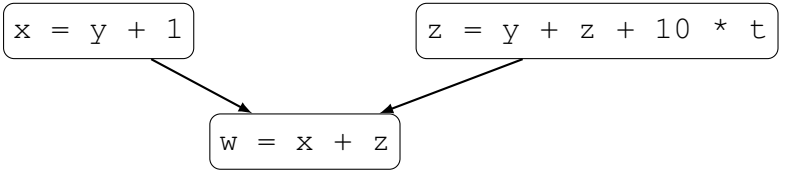
Analyser le potentiel parallèle : le graphe de précédence

```
❶ x := y + 1
❷ z := y + z + 10 * t
❸ w := x + z
```

L'obtention du résultat à partir de x, y et z nécessite **plusieurs étapes** dont certaines peuvent **s'exécuter en parallèle**.

Graphe de précédence

- les nœuds sont les **opérations à réaliser** pour résoudre le problème ;
- les **arcs orientés** sont les **contraintes de précédence** entre les opérations.



⟹On obtient un **ordre partiel** et les **opérations non ordonnées** par cet ordre partiel sont :

▷ **indépendantes**

▷ capables de s'exécuter **en parallèle**.

Sur l'exemple :

▷ ❶ *et* ❷ *peuvent s'exécuter en parallèle ;*

▷ *par contre* ❸ *doit attendre la fin de* ❶ *et* ❷ *avant de débuter.*

Le **graphe de précédence** donne une **analyse statique** du **parallélisme fonctionnel** exploitable :

- la **longueur de son plus long chemin** donne le nombre d'opérations **nécessairement séquentielles** ;
- la **largeur du graphe** donne le **nombre maximum d'opérations exécutables en parallèle** (le degré moyen des nœuds du graphe est également important).

Deux types de source de parallélisme

- le **parallélisme fonctionnel** ou **parallélisme de contrôle** ;
- le **parallélisme de données**.

Parallélisme fonctionnel

Il correspond à :

- découper un problème en **tâches** (opérations élémentaires) ;
- indiquer **l'ordonnancement** de ces tâches (graphe de précédence).

Exemple : produit itératif de n éléments

```
P := a(0) ;
Pour i in 1 .. n - 1 faire
    P := P * a(i) ;
```

P est le produit du premier élément avec le produit des $n-1$ éléments.

Analyse :

- le temps d'exécution est linéaire en n, soit $O(n)$;
- son **graphe de précédence** est une **chaîne** : il ne peut être parallélisable.

Pourtant il est **facile** de concevoir un **algorithme parallèle** à l'aide de n processus en $O(log_2(n))$.

Mais ici, on exploite une **propriété de la multiplication** : son **associativité**.

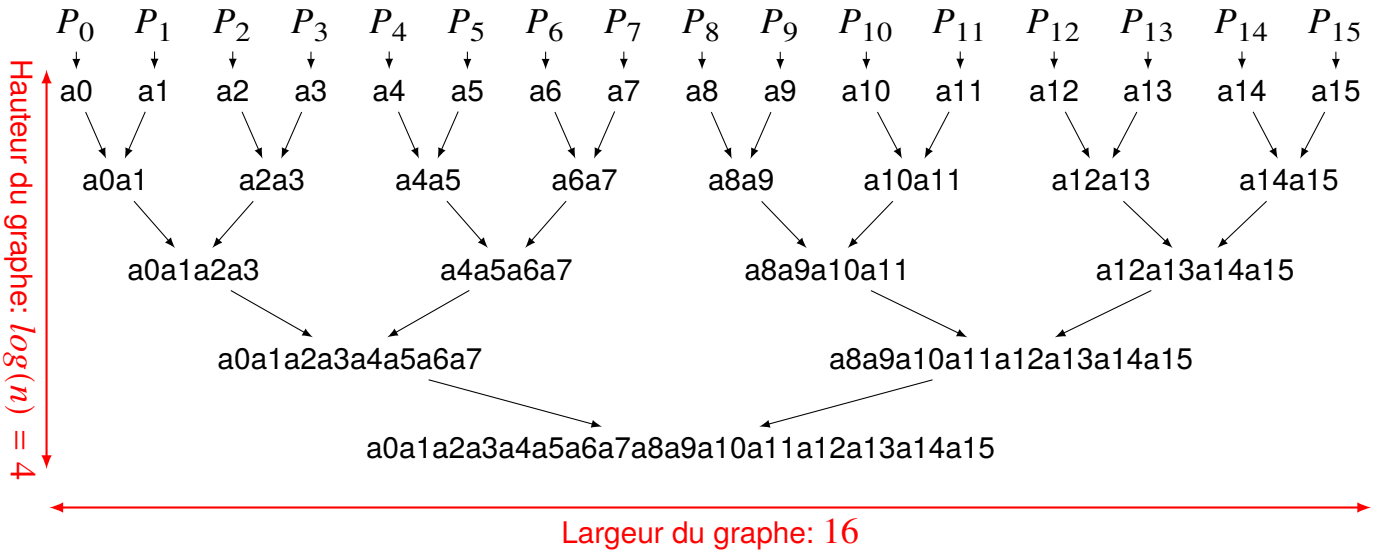




Exemple : produit itératif de n éléments

```
P := a(0) ;
Pour i in 1 .. n - 1 faire
    P := P * a(i) ;
```

P est le produit du premier élément avec le produit des $n-1$ éléments.



Ici, l'**algorithme parallèle** à l'aide de n processus est en $O(log_2(n))$

Syntaxe normale des algorithmes parallèles avec deux instructions pour décrire les opérations paralléles :

▷ Si plusieurs étapes sont faites en paralléle, on écrit :

```
faire étapes i à j en parallèle
   étape i ;
   ...
   étape j ;
fin faire
```

ou avec le « co » pour concurrent :

```
co étape 1 ;              co [i = 0 to n-1] {
//   ...                       a[i] = 0 ; b[i] = 0;
// étape j ;              }
oc
```

▷ Si plusieurs processeurs doivent exécuter le même type d'opérations en parallèle, on écrit :

```
Pour i := 1 à n faire en parallèle
S = { r, s, t, ... } /* S est l'ensemble des données */

opérations réalisées par Pi
fin pour
```





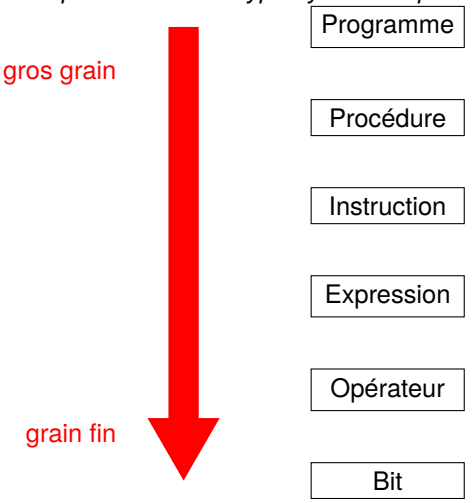

Exploitation (automatique) du Parallélisme





Définition

Le «*grain*» correspond à la taille moyenne des tâches élémentaires.
Le choix du grain lié à l'architecture de la machine (quel type de grain peut être exploité efficacement ?).
Si le grain = programme alors on exploite du parallélisme de type système réparti.



Le «*degré*» de parallélisme reflète le **nombre de processeurs** pouvant être utilisés (degré 1 pour les parties purement séquentielles).
Il peut varier dans les différentes parties des programmes : *degré maximal, minimal, moyen d'un programme*.
Le degré maximal constitue une **borne supérieure** au nombre de processeur qu'on utilisera : avec ce nombre exécution rapide mais efficacité médiocre
en effet si le degré max > degré moyen alors des processeurs seront inactifs pendant une partie de l'exécution du programme

Parallélisme de données

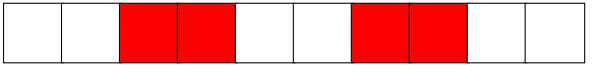
- ▷ les données sont souvent plus nombreuses que les processeurs.
- ▷ la régularité de structures de données permet de distribuer de façon régulière ces données sur les différents processeurs : par exemple pour une matrice $n * n$ sur p processeurs :
  - ◇ chaque processeur possède n/p lignes ;
  - ◇ le processeur est dit **propriétaire** de ces lignes ;
  - ◇ il est **responsable** de la réalisation de toutes les tâches les concernant.

Les **schémas de distribution** les plus classiques sont :

- ☐ répartition par **bloc** :

- ☐ répartition **cyclique** :

- ☐ répartition par **blocs cycliques** :







Exécution synchrone

Le parallélisme de données s'adapte facilement sur un modèle SIMD avec une grande importance donnée à la **synchronisation**.

Exemple :

```
pour i = 0 à n - 1 faire en parallèle
   a[i] = 2.i
pour i = 0 à n - 1 faire en parallèle
   b[i] = a [i] + a[n - 1 - i]
```

Répartition par bloc :

- ▷ `b[0]` est réalisé sur P_0 et nécessite : `a[0]` et `a[n - 1]`
- ▷ **a[0]** calculé sur P_0 mais `a[n-1]` sur P_{n-1}
- ▷ **attention** : pour `b[0]` certaines valeurs de a doivent être disponibles
- ▷ la **synchronisation** est **assurée** par l'utilisation du **modèle SIMD**.

Utilisation de machine du modèle MIMD

Il est possible d'exploiter le **parallélisme de données** sur des machines MIMD :

- ▷ il faut veiller à ce que les processeurs travaillent de **manière coordonnée**

Dans l'exemple, il suffit d'attendre que tous les processeurs aient fini de calculer les valeurs du vecteur `a` avant que l'un d'entre eux ne commence le calcul de `b`.

On **synchronise** donc l'ensemble des processeurs **entre les deux boucles**.

Le parallélisme de données sur des machines MIMD implique un travail de synchronisation de la part du programmeur.

Dépendance des données

Reprenons l'exemple précédent dans une forme *condensée* :

```
a[0]  = 0
sp[0] = a[0]
pour i = 1 à n - 1 faire
     a[i]  = 2.i
     sp[i] = a[i] + sp [i - 1]
```

Le calcul des éléments du vecteur sp ne peut être fait en parallèle...mais pourquoi ?

Notions de variables lus et modifiées pour une instruction S

- ▷ $L(S)$ ensemble des variables utilisées en **lecture** ;
- ▷ $M(S)$ ensemble des variables qui subissent une **modification** (une affectation).

Condition de Bernstein pour assurer l'indépendance en deux instructions séquentielles S_1 & S_2

- □ $M(S_1) \cap L(S_2) = \emptyset$: dépendance vraie, ou RAW, «S_2*Read-After-*S_1*Write*»
- □ $L(S_1) \cap M(S_2) = \emptyset$: anti-dépendance, ou WAR, «S_2*Write-After-*S_1*Read*»
- □ $M(S_1) \cap M(S_2) = \emptyset$: dépendance de sortie, ou WAW, «S_2*Write-After-*S_1*Write*»

*Ces 3 équations doivent être **vérifiées** pour pouvoir réaliser S_1 et S_2 en **parallèle**.*

Il est possible de définir :

- ▷ S comme une **séquence d'instructions** au lieu d'une seule instruction ;
- ▷ les ensembles L et M en les calculant sur les instructions de chaque séquence.

La **condition de Bernstein** exprime la possibilité de calculer en **parallèle** les deux séquences d'instructions S_1 et S_2.

Dépendance des données

Exemple :

```
a[0]  = 0
sp[0] = a[0]
pour i = 1 à n - 1 faire
     a[i]  = 2.i
     sp[i] = a[i] + sp [i - 1]
```

Le calcul des éléments du vecteur sp ne peut être fait en parallèle...mais pourquoi ?

Condition de Bernstein pour assurer l'indépendance en deux instructions séquentielles S_1 & S_2

- $M(S_1) \cap L(S_2) = \emptyset$: dépendance vraie
- $L(S_1) \cap M(S_2) = \emptyset$: anti-dépendance
- $M(S_1) \cap M(S_2) = \emptyset$: dépendance de sortie

*Ces 3 équations doivent être **vérifiées** pour pouvoir réaliser S_1 et S_2 en **parallèle**.*

Exemple

On numérote S_1, et S_2 les instructions de la boucle :

xterm

```
a[0]  = 0
sp[0] = a[0]
pour i = 1 à n - 1 faire
   S1: a[i] = 2.i
   S2: sp[i] = a[i] + sp [i - 1]
```

- $M(S_1) = \{a[i]\}$
- $L(S_2) = \{a[i]\}$

} *alors* $M(S_1) \cap L(S_2) \neq \emptyset$, *on a une* «dépendance vraie» *ou RAW.*




Dépendance des données

Exemple :

```
a[0]  = 0
sp[0] = a[0]
pour i = 1 à n - 1 faire
     a[i]  = 2.i
     sp[i] = a[i] + sp [i - 1]
```

Le calcul des éléments du vecteur sp ne peut être fait en parallèle...mais pourquoi ?

Condition de Bernstein pour assurer l'indépendance en deux instructions séquentielles S_1 & S_2

- $M(S_1) \cap L(S_2) = \emptyset$: dépendance vraie
- $L(S_1) \cap M(S_2) = \emptyset$: anti-dépendance
- $M(S_1) \cap M(S_2) = \emptyset$: dépendance de sortie

*Ces 3 équations doivent être **vérifiées** pour pouvoir réaliser S_1 et S_2 en **parallèle**.*

Exemple

On note S_1 les instructions de la boucle et S_2 les instructions de l'occurrence suivante :

xterm

```
a[0] = 0
sp[0] = a[0]
pour i = 1 à n - 1 faire        Soit j une occurrence de la boucle
   S1: a[j] = 2.j
         sp[j] = a[j] + sp [j - 1]
   S2: a[j+1] = 2.(j+1)
         sp[j+1] = a[j+1] + sp [j]
```

«*Dépendance vraie*» entre les instances de l'instruction S_1 (pour j) et S_2 (pour $j+1$) dans la boucle $M(S_1) \cap L(S_2) = \{sp[j]\} \neq \emptyset \Longrightarrow$ Condition de Bernstein **non vérifiée**




Modifications pour tirer parti du parallélisme

xterm
```
a[0] = 0
sp[0] = a[0]
pour i = 0 à n - 1 faire
   a[i] = 2.i
   sp[i] = a[i] + sp [i - 1]
```

Peut être réécrit en :

xterm
```
a[0] = 0
sp[0] = a[0]
pour i = 0 à n - 1 faire
   a[i] = 2.i
pour i = 0 à n - 1 faire
   sp[i] = a[i] + sp [i - 1]
```

Et enfin :

xterm
```
a[0] = 0
sp[0] = a[0]
pour i = 1 à n-1 faire_en_parallèle
   a[i] = 2.i

pour i = 1 à n - 1 faire
   sp[i] = a[i] + sp [i - 1]
```

Au final :

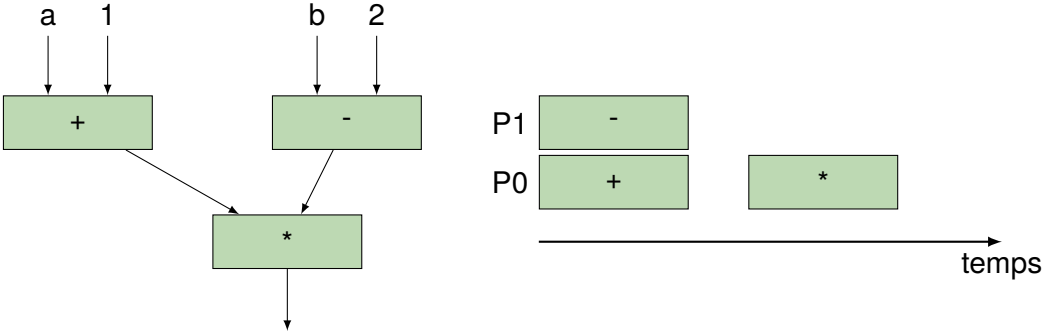
- ▷ Le **première** boucle peut être **exécutée en parallèle !**
- ▷ La **seconde** boucle établie une **dépendance temporelle** entre une occurrence de la boucle et sa suivante :
  $\Longrightarrow$**pas d'exécution parallèle possible !**




Exploitation du parallélisme

- ▷ 1ère phase : découpage en tâches ;
- ▷ 2ème phase : étude des tâches pour déterminer celles qui peuvent s'exécuter en parallèle et celles qui doivent s'exécuter séquentiellement ;

Exemple :
graphe de $(a + 1) * (b - 2)$



Choix du grain

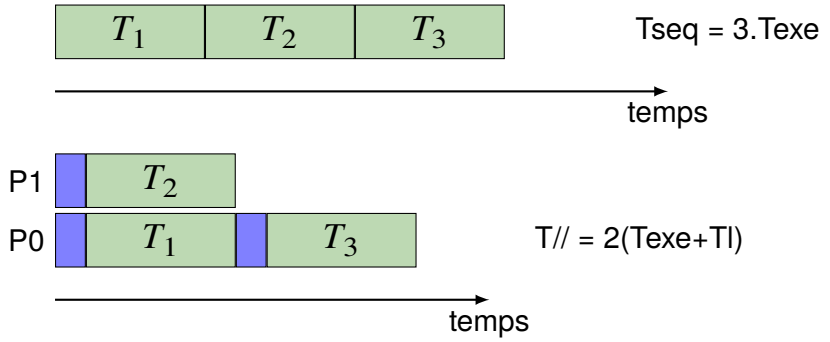
La taille des tâches, le «*grain*» obtenu lors du découpage est important pour l'accélération.

Le grain doit être adapté à la machine sous-jacente :

- ◇ si le **lancement d'une tâche** prend un temps important, on ralenti l'exécution en définissant un grand nombre de tâches
- ◇ les **temps d'exécution des tâches** tournant en parallèle doivent être **voisins**.

Soient 3 tâches T_1, T_2 et T_3 avec T_3 dépend de T_1 et T_2

- Texe : temps d'exécution de la tâche
- Tl : temps de lancement de la tâche



Si Tl > Texe / 2 alors T// > Tseq !

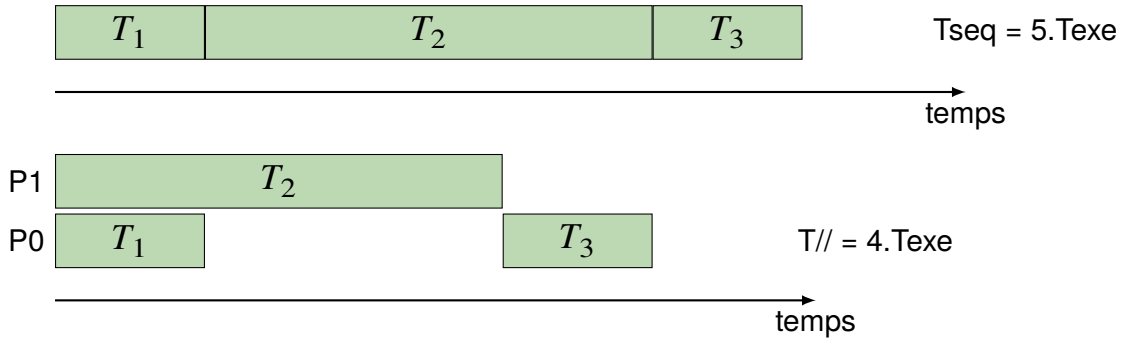
Que faire ?

- ▷ Soit on diminue Tl en changeant de machine //
- ▷ Soit on augmente le **grain** des tâches en découpant le programme autrement : si Tl est **négligeable** devant Texe, **l'accélération** et **l'efficacité** sont **meilleures**.




Supposons :

- Tl est négligeable devant Texe
- T2 prend 3.Texe



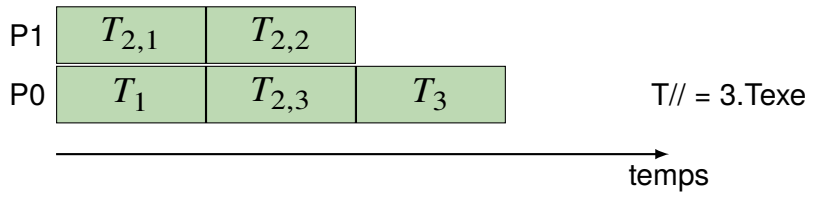
- Accélération = 5/4
- Efficacité = 5/8

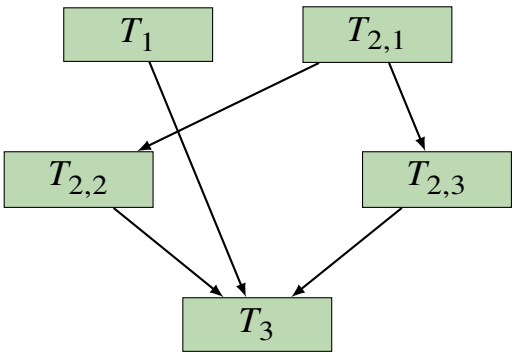
Ce qui n'est pas très bon !




Solution 1 :

– découper T2 en 3 sous-tâches // (T2,1 ; T2,2 ; T2,3) de durée Texe
– T2,1 précède les deux autres qui peuvent être exécutées **en parallèle**.





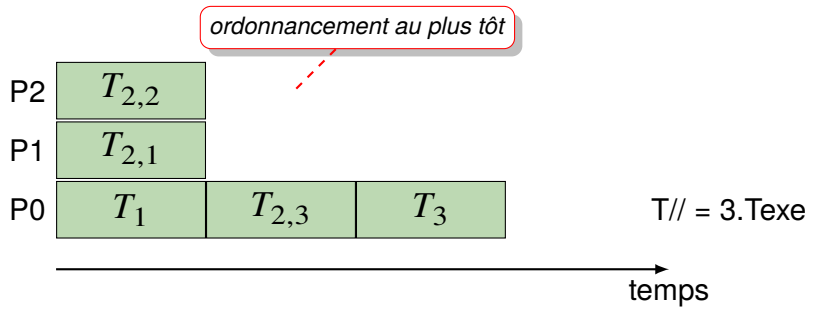
▷ Accélération = 5/3

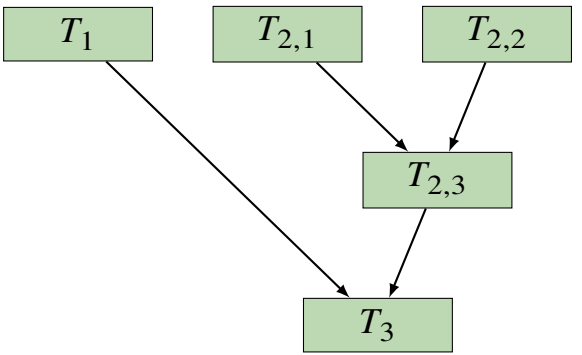
▷ Efficacité = 5/6

Ce qui est meilleur !

Solution 2 :

– découper T2 en 3 sous-tâches // (T2,1 ; T2,2 ; T2,3) de durée Texe
– T2,1 et T2,2 indépendantes et précédant T2,3.





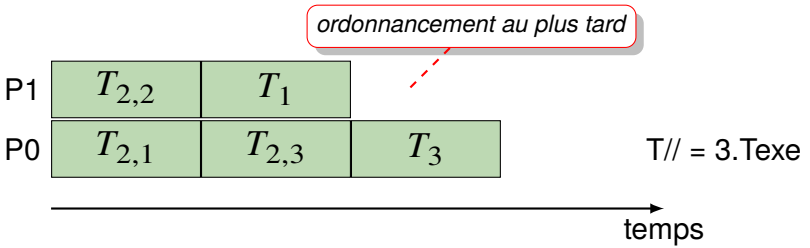
▷ Accélération = 5/3

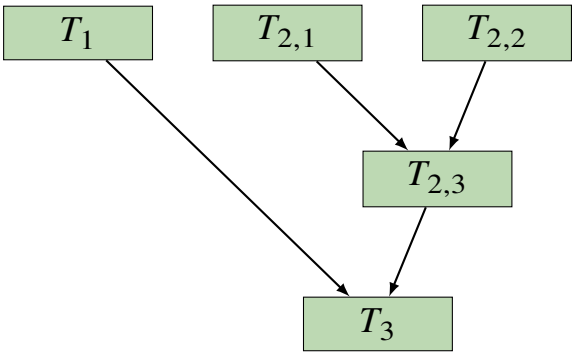
▷ Efficacité = 5/9

On notera que l'arbre de précédence exprimant les dépendances temporelles entre les différentes tâches est différent de celui du transparent précédent : un nouveau découpage de T2 a été trouvé.

Solution 2 :

– découper T2 en 3 sous-tâches // (T2,1 ; T2,2 ; T2,3) de durée Texe
– T2,1 et T2,2 indépendantes et précédant T2,3.





▷ Accélération = 5/3

▷ Efficacité = 5/6

Le coût des communications

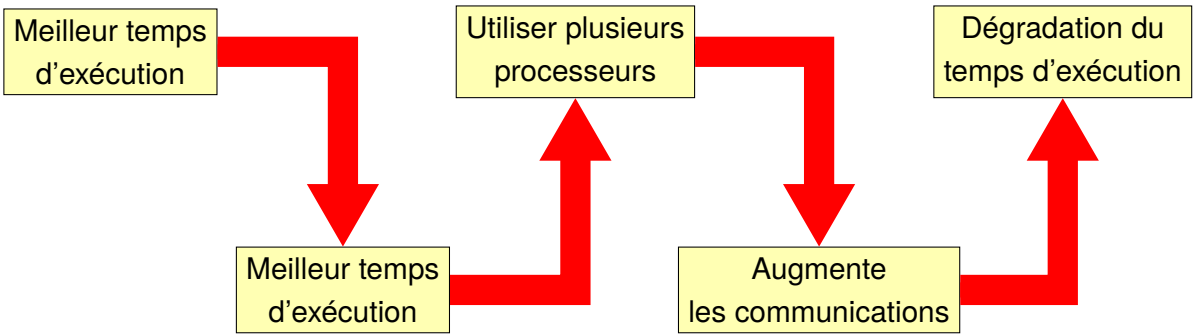
Sur une machine à mémoire distribuée, le coût des communications est important.

Si deux tâches dépendantes sont sur des processeurs distincts, le résultat de la 1ère doit être émis vers la 2ème.

Le meilleur temps d'exécution correspond à la répartition des tâches de manière à :

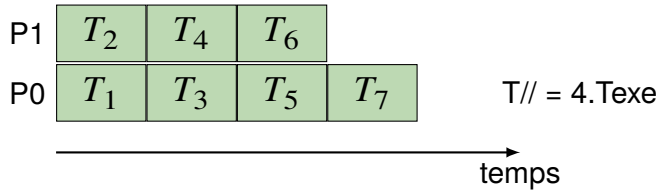
- ▷ minimiser les temps de calcul
- ▷ minimiser les temps d'attente
- ▷ minimiser les communications

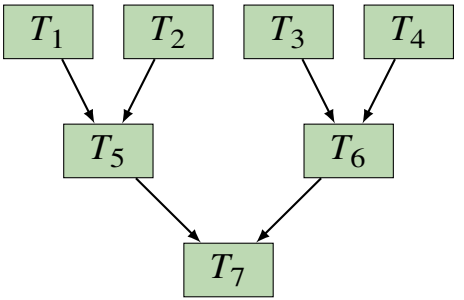
Importance du découpage et du placement des tâche



Placement des tâches

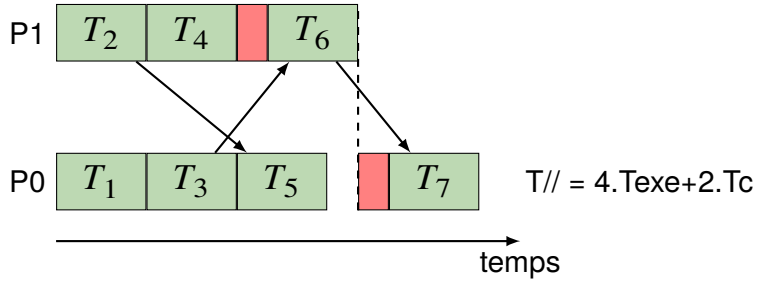
Exemple : 7 tâches de même durée Texe, sur le schéma : placement optimal au niveau du temps de calcul

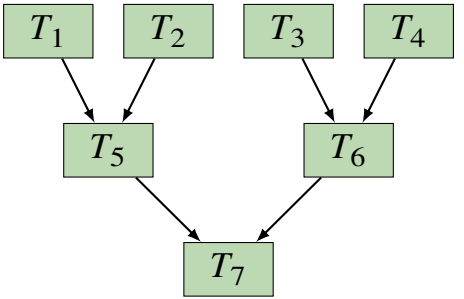




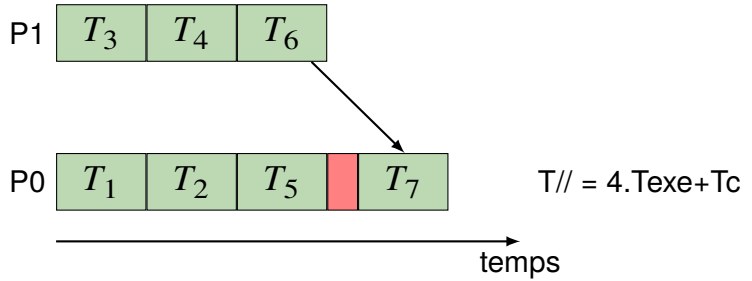
Temps de communication : Tc < Texe

Communication par des circuits spécialisés

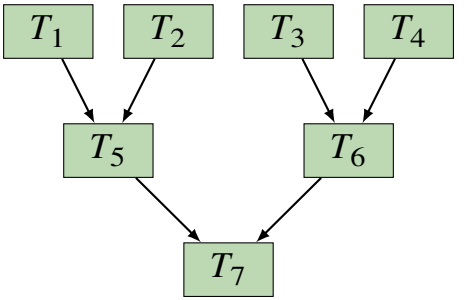




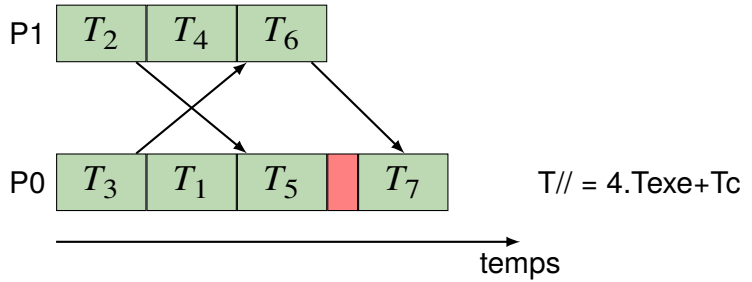
Optimisation : Déplacement des tâches entre les processeurs pour diminuer les communications



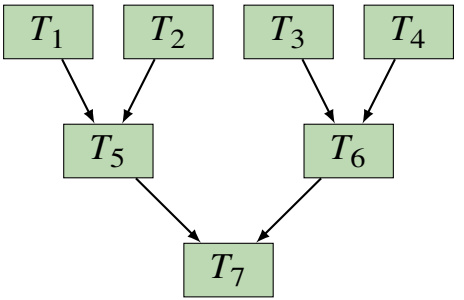
T// = 4.Texe+Tc



Réordonnancement des tâches sur un même processeur pour les faire communiquer **pendant** qu'ils calculent



T// = 4.Texe+Tc





Bilan sur le processus de parallélisation

- Le choix des tâches définit le **degré** de parallélisme :

  plus le nombre de tâches ↗, plus le grain ↘

- La **taille d'une tâche** doit être **suffisamment grande** pour que **Tl** soit négligeable devant **Texe** ;
- La **taille des tâches** qui s'exécutent simultanément doit être «voisines» ou «identiques» ;
- L'ordonnancement des différentes tâches doit être fait de manière choisie : probablement de manière **empirique** par évaluation sur différentes exécutions ;
- Il faut trouver un **placement efficace** des différentes tâches sur les nœuds de la machine ;
- Sur une machine à **mémoire distribuée**, il faut également tenir compte des **communications** (encore plus sur un cluster de station de travail) :

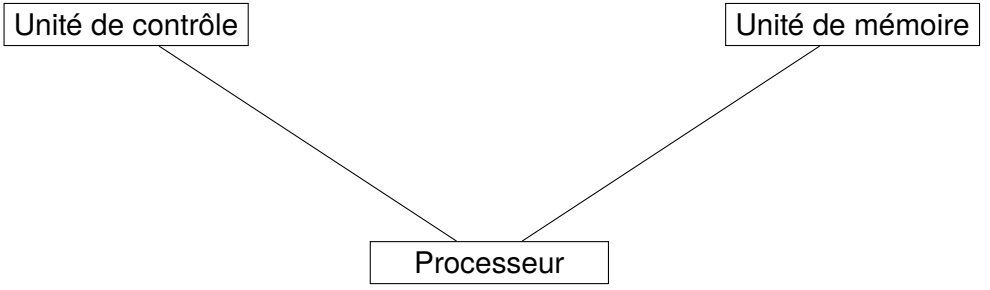
  $\Longrightarrow$ **recouvrement calcul/communication**



Et par rapport aux architectures parallèles ?



- un flot d'instructions ;
- un flot de données ;



- ▷ À chaque étape de calcul, l'unité de contrôle produit une instruction qui opère sur une donnée obtenue à partir de l'unité mémoire.
- ▷ Il n'y a qu'un seul processeur⟹**modèle séquentiel**.

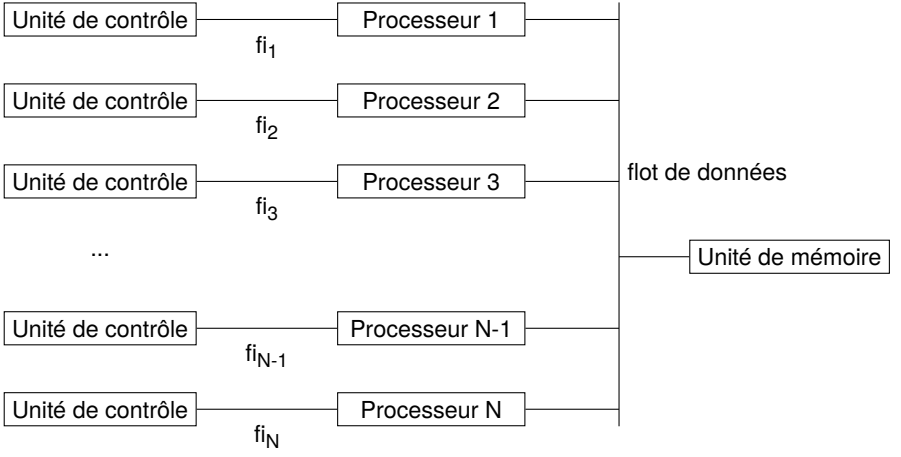


A chaque étape, une donnée est :

- fournie par l'unité mémoire ;
- traitée parallèlement par les N processeurs qui exécutent chacun une instruction spécifique.

On a plusieurs **flots d'instructions**, fi, opérant sur un seul **flot de données**.

On a N processeurs, $N > 1$, chacun ayant son **unité de contrôle**, partageant une unique **unité mémoire**.



Exemple

On veut savoir si un nombre z est premier.

On peut associer à chaque processeur un diviseur potentiel de E (nombres entre z et z-1).

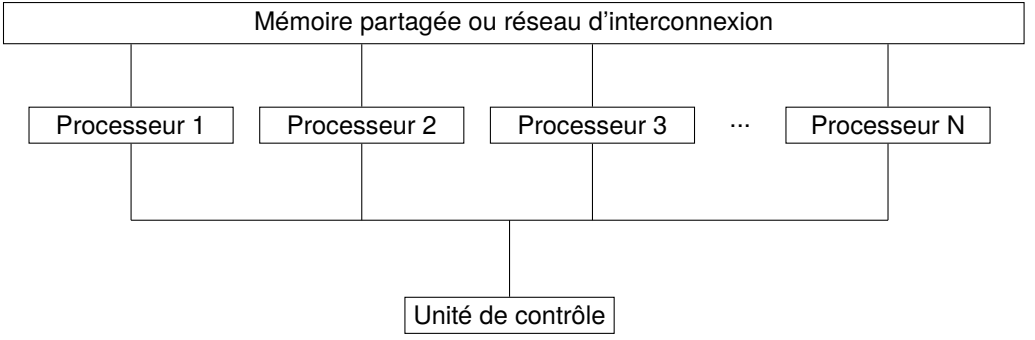
Un processeur :

- reçoit z ;
- fait la division par son le diviseur potentiel ;
- répond par « oui » ou « non » dans une zone mémoire fixe, connue par tous les processeurs.

Il n'existe pas de calculateur fonctionnant avec ce modèle !



On dispose de N processeurs, tous identiques ayant chacun une mémoire locale où on peut stocker des données et des instructions.



On suppose la mémoire locale intégrée au processeur.

Tous les processeurs opèrent sous le contrôle d'un unique flot d'instruction (ceci est équivalent à avoir dans chaque mémoire locale une copie d'un unique programme).

Les processeurs opèrent de manière **synchrone** (horloge globale) sur les N flots de données (à chaque pas de temps, ils exécutent tous la même instruction).

À certaines étapes, certains processeurs peuvent rester inactifs, en fait ils exécutent « une instruction d'attente » ayant la durée voulue (nécessité de conserver une totale synchronisation entre tous les processeurs.

Les processeurs communiquent (échangent des données) pour coordonner leurs résultats. Ceci se fait de deux manières différentes :

– **mémoire partagée** SM SIMD (shared memory) ;
– **réseau d'interconnexion** (interconnection network).



Le modèle est connu sous le nom de modèle **PRAM**, «*Parallel Random Access Machine*».

Les processeurs utilisent la mémoire la **mémoire partagée** de la manière suivante :
« si le processeur i veut communiquer une donnée avec le processeur j, alors i écrit le nombre en mémoire partagée dans une zone connue par j, puis j va lire le nombre »

Quand les N processeurs vont faire ces actions en même temps, il y aura des problèmes de concurrence d'accès.

On aura quatre sous classes du modèle SM SIMD :
- ▷ EREW, Exclusive Read Exclusive Write ;
- ▷ CREW, Concurrent Read Exclusive Write ;
- ▷ ERCW, Exclusive Read Concurrent Write ;
- ▷ CRCW, Concurrent Read Concurrent Write.

Puissance du modèle ↓

A priori les accès concurrents en **lecture** ne posent pas de problème.

Par contre, en **écriture**, il faut une règle pour trancher le problème :
a. le processeur ayant le plus petit numéro parmi ceux voulant accéder en écriture sera prioritaire ;
b. l'écriture est interdite sauf si tous les processeurs écrivent la même valeur ;
c. on écrit la somme des valeurs que les processeurs veulent écrire.



Le modèle SM SIMD : Exemple 1

On considère un fichier contenant n enregistrements distincts rangés dans un ordre quelconque.
On veut savoir si un enregistrement x donné est présent dans le fichier ou pas.
Dans un modèle SISD, (machine séquentielle), ceci prend au pire des cas n comparaisons et $n/2$ en moyenne.
On considère le modèle EREW SM SIMD avec N processeurs et $N \leq n$.
On note les processeurs P_1, ... , P_N.
Pour commencer, tous les processeurs doivent connaître x.
Ceci se fait par une opération de « broadcasting » (one to all) à partir de P_1.

- étape 1 : P_1 lit x et le communique à P_2 : $P_1 \overset{x}{\rightarrow} P_2$
- étape 2 :
  - $P_1 \overset{x}{\rightarrow} P_3$
  - $P_2 \overset{x}{\rightarrow} P_4$ en parallèle
- étape 3 :
  - $P_1 \overset{x}{\rightarrow} P_5$
  - $P_2 \overset{x}{\rightarrow} P_6$
  - $P_3 \overset{x}{\rightarrow} P_7$
  - $P_4 \overset{x}{\rightarrow} P_8$ en parallèle
- *etc.*
- étape t : les processeurs P_i, $1 \leq i \leq 2^t$ connaissent x

On suppose que le fichier est découpé en N sous-fichiers de taille n/N et que chaque processeur P_i recherche x dans le $i^{ème}$ sous fichier.
Les N recherches se font en parallèle donc il faut au pire des cas n/N étapes de comparaison.
La réponse finale est fournie (si elle est positive) par l'unique processeur ayant trouvé x dans son sous- fichier.

il faut donc au plus une étape pour fournir le résultat dans la mémoire partagée (dans une variable prédéfinie initialisée à Faux).
Soit au pire des cas, on a $log_2(N) + n/N + 1$.

Amélioration ?

On regarde ce qui se passe avec une approche « moyenne », *i.e.* on arrête dès que x est trouvé.

On considère une variable booléenne partagée F ($F \in P_1$) et initialisée à faux.

Quand un processeur trouve x, il met F à vraie.

À chaque étape, tous les processeurs recevront F et s'arrêtent si F est vraie (point de synchronisation forte : coûteux sauf si on a un très bon réseau d'interconnexion).

En moyenne on a :

$\frac{1}{2}[log_2(N) + \frac{n}{N}(1 + log_2(N)) + 1 + log_2(N)] = log_2(N) + \frac{n/N+1}{2}(1 + log_2(N))$

$\Longrightarrow$Ce qui est moins bon !

Pour mieux exploiter cet arrêt possible, on considère le modèle CREW SM SIMD :
$\Longrightarrow$Les **lectures simultanées** vont permettre de remplacer des termes $log_2(N)$ par 1.

$\Longrightarrow$On aura alors $n/N + 2$ étapes comme dans le pire des cas.

Remarque : si x est présent plusieurs fois, il faut un modèle CRCW pour avoir le même résultat.

On veut pouvoir simuler des accès concurrents à l'aide du modèle EREW.

On a N processeurs $P_1, ..., P_N$ qui veulent simultanément **lire** le contenu d'une variable A, ou **écrire** dans A.

- ▷ **lecture** : $log_2(N)$ étapes sont nécessaires
- ▷ **écriture** : on appelle a_i , $1 \leq i \leq N$, la valeur que veut écrire le processeur P_i.

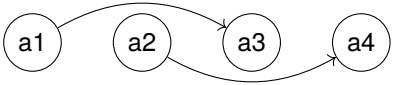
On suppose que les N processeurs pourront écrire dans A que si tous les a_i sont égaux.

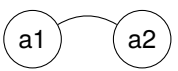
1. Pour tout i, $1 \leq i \leq N/2$ faire simultanément :
   Si $a_i = a_{i+N/2}$ alors P_i met à vrai la variable b_i sinon P_i met à faux la variable b_i
2. Pour tout i, $1 \leq i \leq N/4$ faire simultanément :
   Si b_i et $b_{i+N/4}$ vraies et $a_i = a_{i+N/4}$ alors P_i met à vrai la variable b_i sinon P_i met à faux la variable b_i

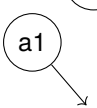
On itère en divisant l'intervalle par 2 à chaque étape et après $log_2(N)$ étapes P_1 sait si tous les a_i sont égaux.
$\Longrightarrow$ Si oui P_1 écrit a_1 dans A et sinon il ne fait rien (A reste à faux).

Exemple

$N = 4$

- ▷ 1ère étape : comparaisons 

- ▷ 2ème étape : comparaison 

- ▷ 3ème étape : écriture 

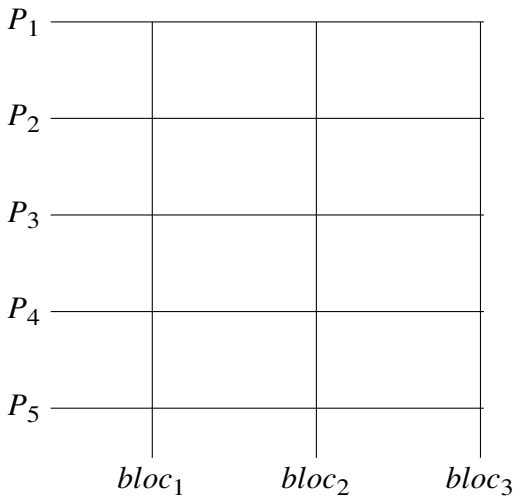
Ce modèle est très puissant mais il est peut réaliste à cause de la complexité des circuits d'accès à la mémoire.

Si la mémoire possède M zones élémentaires, le coût du circuit permettant un calcul d'adresse pour un processeur varie comme $f(M)$ ou f est croissante.

Dans le cas de N processeurs partageant la mémoire, le coût du circuit global varie comme $N * f(M)$.

$\Longrightarrow$Ceci est **irréaliste** pour N et M grands.

Une solution pour diminuer ce coût est de diviser la mémoire en R blocs de taille M/R chacun.



La **concurrence** se fait au niveau du bloc.

Un processeur quelconque peut accéder à un bloc quelconque.

Il y a $N * R$ «*switches*» : un à chaque intersection bus mémoire/bus processeur.

L'accès se fait par une logique « ligne-colonne » à travers les switches.

Chaque bloc de mémoire dispose d'un circuit d'adressage interne (M/R possibilités)

Le coût est de : $R * f(M/R) + (N * R) *$ « coût d'un switch »
Le coût d'un switch est négligeable.

$\Longrightarrow$Le modèle est **moins puissant** à cause de l'accès au niveau des blocs.

On veut étendre l'idée précédente :
Les M éléments de mémoire sont distribués sur les N processeurs, chacun ayant localement M/N mémoire.

Chaque **paire de processeurs** est connectée par un lien bidirectionnel et à chaque étape un processeur quelconque P_i peut :

- ▷ recevoir une donnée d'un processeur P_j
- ▷ envoyer un autre message à P_k (k peut être égal à j).

Chaque processeur doit avoir un circuit d'adressage de coût $f(N-1)$ lui permettant de sélectionner n'importe lequel des autres processeurs.

Il doit avoir un circuit d'adressage de coût $f(M/N)$ pour permettre un accès à sa propre mémoire.

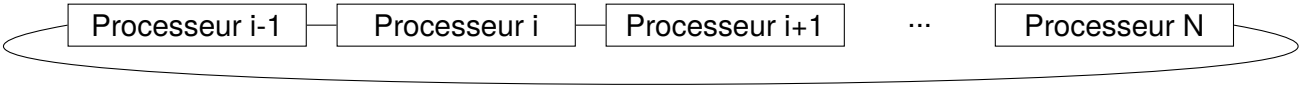
On aura en tout :

$$N * (f(N-1) + f(M/N))$$

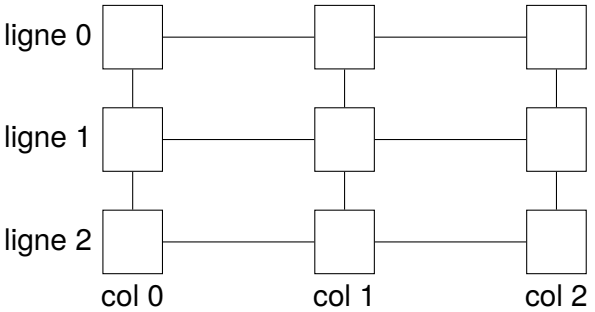
Le coût est **trop important** mais le modèle est **plus puissant** que celui avec découpage en blocs de la mémoire.

$\Longrightarrow$On a besoin de réseaux « **faiblement couplés** » par opposition au **graphe complet**.

La chaîne «*1D*»



La grille «*2D*»

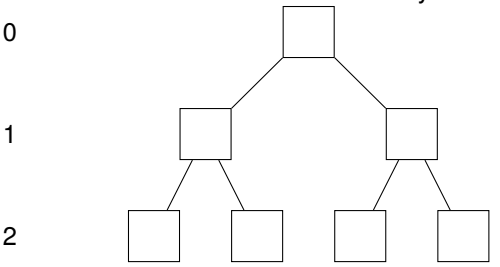


Processeur $P_{i,j}$:

- ligne i ;
- colonne j ;

L'arbre binaire

On a un arbre binaire complet avec d niveaux numérotés de 0 à $d-1$ et il y a $N = 2d-1$ processeurs.



Le «*Perfect Shuffle*» ou mélange parfait

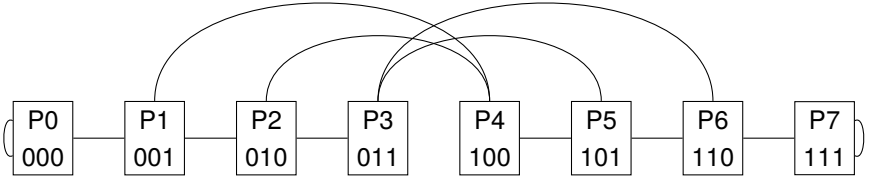
On a N processeurs numérotés P_0, ... , P_{N-1}, avec $N = 2^P$

P_i est connecté à P_j si et seulement si :

▷ $j = 2 * i$ pour $0 \leq i \leq N/2-1$;

▷ $j = 2 * i + 1-N$ pour $N/2 \leq i \leq N-1$;

$\Longrightarrow$L'écriture binaire de j se déduit de celle de i par un décalage circulaire à gauche : liens « shuffle ».



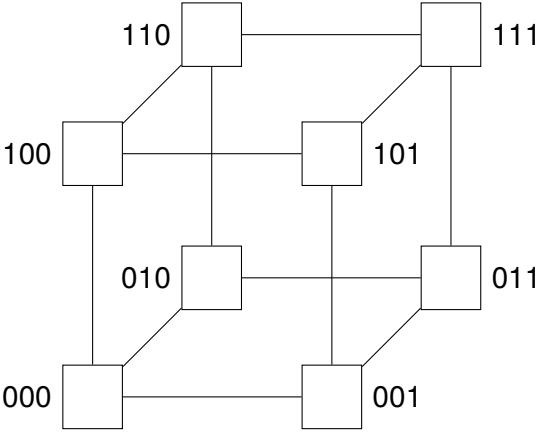
On rajoute des liens d'échanges entre tout processeur ayant un numéro pair et son suivant.

On a le « shuffle exchange ».

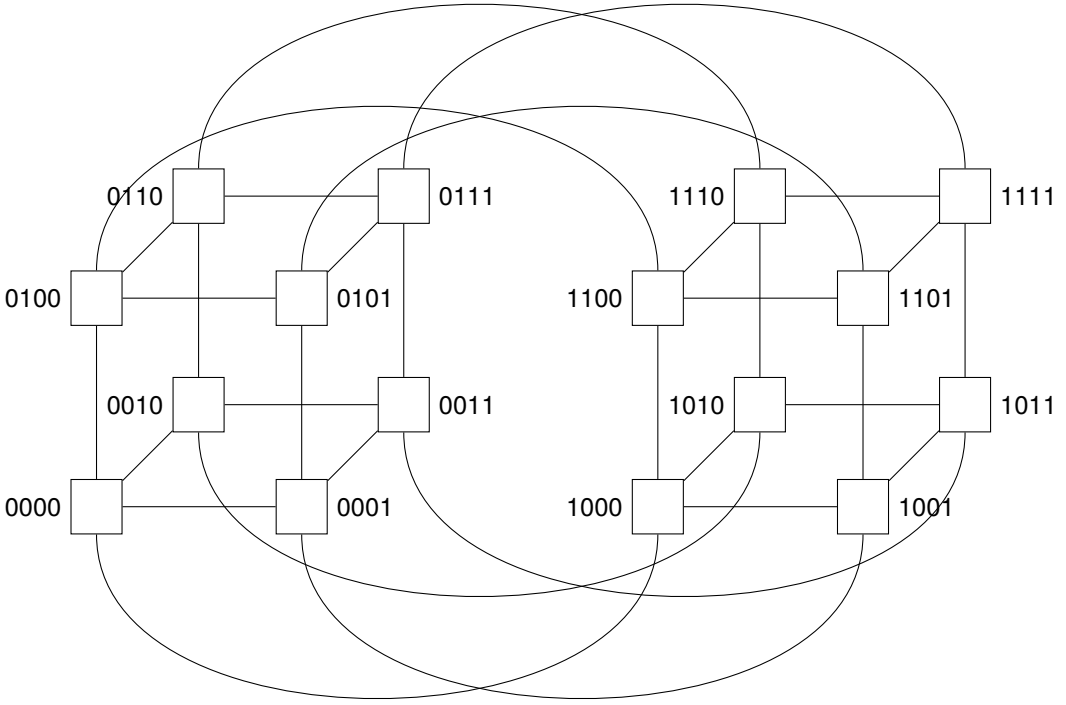
Ce type de réseau est bien adapté pour le calcul de la **transformée de Fourier rapide** ou FFT en parallèle.

Le cube

On considère $N = 2^q$ processeurs numérotés $P_0, P_1, ..., P_{N-1}$ ($q \geq 1$)
Un cube de dimension q (hypercube) est obtenu en connectant un sommet à exactement q voisins.
Ce qui donne un degré logarithmique.
Pi est connecté à Pj si et seulement si les écritures binaires de i et de j ne différents que d'un chiffre binaire.




Le cube de degré 4




Exemple sur l'arbre binaire

On veut faire la somme de n nombres $x_1, x_2, ... ,x_n$.

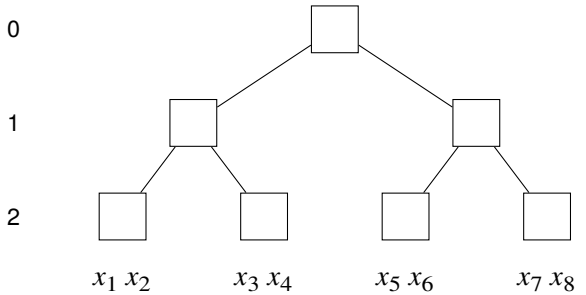
Dans un modèle SISD, il faut n étapes ($n-1$ additions et 1 affectation).

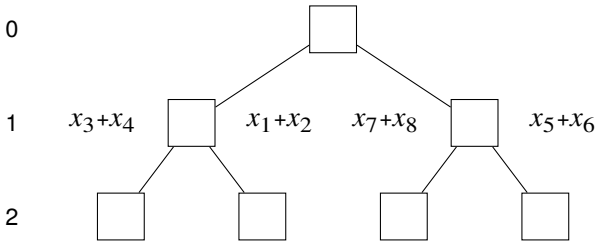
En utilisant un arbre ayant $log_2(n)$ niveaux et $n/2$ feuilles, le calcul peut se faire en $log_2(n)$ étapes.

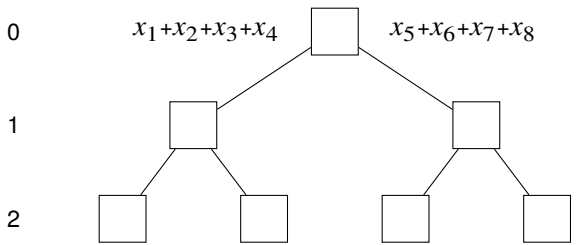
Chaque feuille a au départ 2 nombres, elle fait la somme et l'envoie au père.

Pour les processeurs des autres niveaux, on fait la chose suivante :

- ▷ recevoir une donnée de chacun des deux fils ;
- ▷ faire la comme ;
- ▷ envoyer le résultat au père.







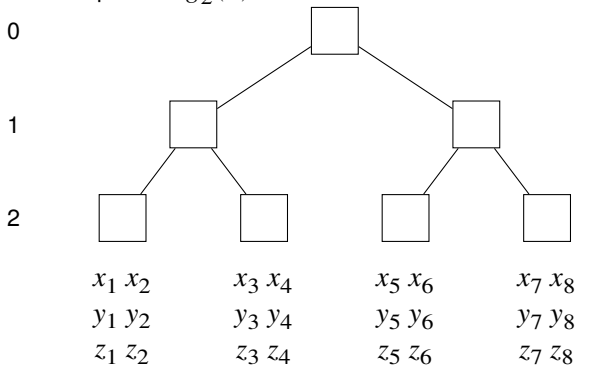
La racine n'a plus qu'à faire la **somme finale** :

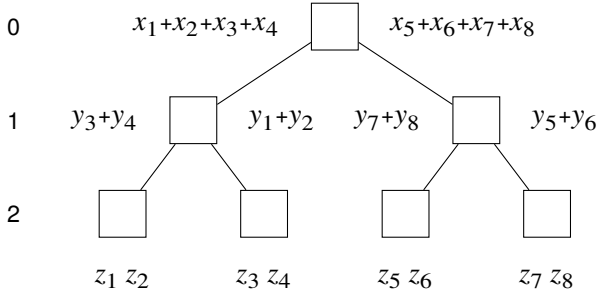
$$x_1+x_2+x_3+x_4 + x_5+x_6+x_7+x_8$$

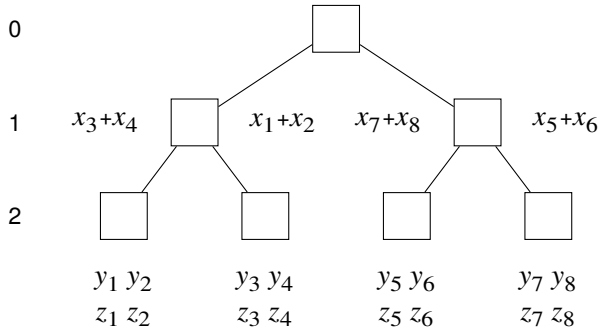
Effet pipeline

Si on veut faire m sommations de n nombres, on « pipeline » les différents calculs.

Le calcul prend $log_2(n) + m - 1$.







La racine fait la première **somme** :

$$x_1+x_2+x_3+x_4 + x_5+x_6+x_7+x_8$$

Puis à l'étape suivante, la seconde somme :

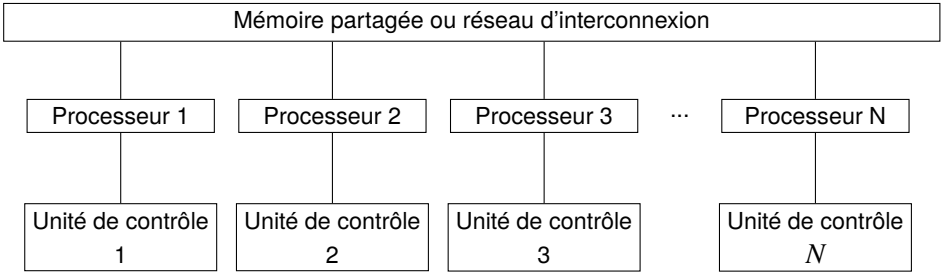
$$y_1+y_2+y_3+y_4 + y_5+y_6+y_7+y_8$$

Puis à l'étape suivante, la troisième somme :

$$z_1+z_2+z_3+z_4 + z_5+z_6+z_7+z_8$$ *etc.*

C'est le modèle **le plus général**.

Il y a N processeurs avec N flots de données différents.



Chaque processeur dispose d'une **mémoire locale**.

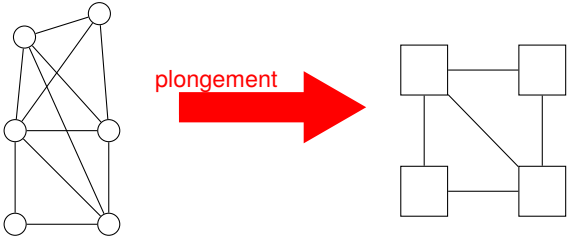
Pour le modèle avec mémoire partagée, on a la même classification qu'en SIMD au niveau de la concurrence des accès.

On a aussi tous les réseaux vus en SIMD : anneau, grille, hypercube, ...

Les algorithmes s'expriment en terme de processus communicants par **SM**, «*Shared Memory*», ou par **transmission de messages**.

On a généralement n processus sur N processeurs avec $N \leq n$

- ▷ **placement des processus** sur les processeurs (problème NP-complet) ;
- ▷ placement **statique** ou **dynamique** ;
- ▷ **équilibrage de la charge** des processus (dynamique ou statique) ;
- ▷ **minimisation** de la **communication** et du **routage**.



Et l'évaluation de nos algorithmes parallèles sur ces modèles ?

On veut des **critères** pour évaluer la **qualité** des algorithmes parallèles.

Les **métriques** classiques sont :

- le temps d'exécution ;
- le nombre de processeurs utilisés ;
- le coût.

Le temps d'exécution :

- le temps entre le début et la fin du calcul ;
- temps écoulé entre le moment où le premier processeur démarre et le moment où le dernier arrête.

Les unités sont :

- celles de calcul : calcul arithmétique ou logique ;
- celles de routage : on doit faire intervenir la distance entre les processeurs dans le schéma de communication dans le réseau.

On notera $t(n)$ le temps d'exécution parallèle dans le pire des cas pour un problème de taille n.

On calcule des bornes inférieure et supérieure : notations (Ω, σ).

On évalue la qualité de l'algorithme par **l'accélération** : «*speedup*».

$$\text{speedup} = \frac{\text{temps d'exécution dans le pire des cas du meilleur algorithme séquentiel}}{\text{temps d'exécution dans le pire des cas de l'algorithme parallèle} \Longrightarrow t(n)}$$

Si on a N processeurs valant $t(n)$ on a un speedup $\leq N$.

On note $p(n)$ le nombre de processeurs pour l'algorithme parallèle traitant le problème de taille n.
C'est un critère économique.

Pour un problème de taille n, le coût d'un algorithme parallèle est :
$c(n) = t(n) * p(n)$
Si tous les processeurs exécutent la même masse de travail, $c(n)$ est le nombre total d'opérations du problème.
Sinon, on a un majorant de ce nombre.

Si une borne inférieure de ce nombre est connue et si elle correspond au coût de l'algorithme séquentielle, on dit que **l'algorithme parallèle** est à **coût optimal**.

Un algorithme parallèle n'est **pas à coût optimal** s'il existe un algorithme séquentiel dont la complexité est **inférieure** à son coût.

Problème dit de la « scalabilité » ou extensibilité

Est-ce que $t(n)$ **diminue** lorsque $p(n)$ **augmente** ?

Sur les exemples

1. Recherche dans un fichier à n éléments avec N processeurs
   Modèle «*CREW SM SIMD*» :
   - $t(n) = o(n/N)$
   - $p(n) = N$ alors $c(n) = o(n) \Longrightarrow$ **coût optimal**

2. Somme de n nombres sur un arbre à $n-1$ processeurs :
   - $t(n) = log_2(n)$
   - $p(n) = n-1$ alors $c(n) = o(n * log_2(n)) \Longrightarrow$ **pas à coût optimal**

   pour m sommes de n nombres avec l'effet pipeline :
   - $t(n) = m-1 + log_2(n)$
   - $p(n) = n-1$ alors $c(n) = o(m.n) \Longrightarrow$ **coût optimal**

Notion d'efficacité

On introduit aussi **l'efficacité** :

$$\text{efficacité} = \frac{\text{speedup}}{p(n)} \leq 1$$