

Cours d'Architecture des Ordinateurs « Parallèles »

Ronan.Keryell@enstb.org

—

Département Informatique

École Nationale Supérieure des Télécommunications de Bretagne

Master Recherche 2^{ème} année Informatique de Rennes 1 — ENSTBr

ISIA — ENSMP

octobre 2006–février 2007

Version 1.15

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

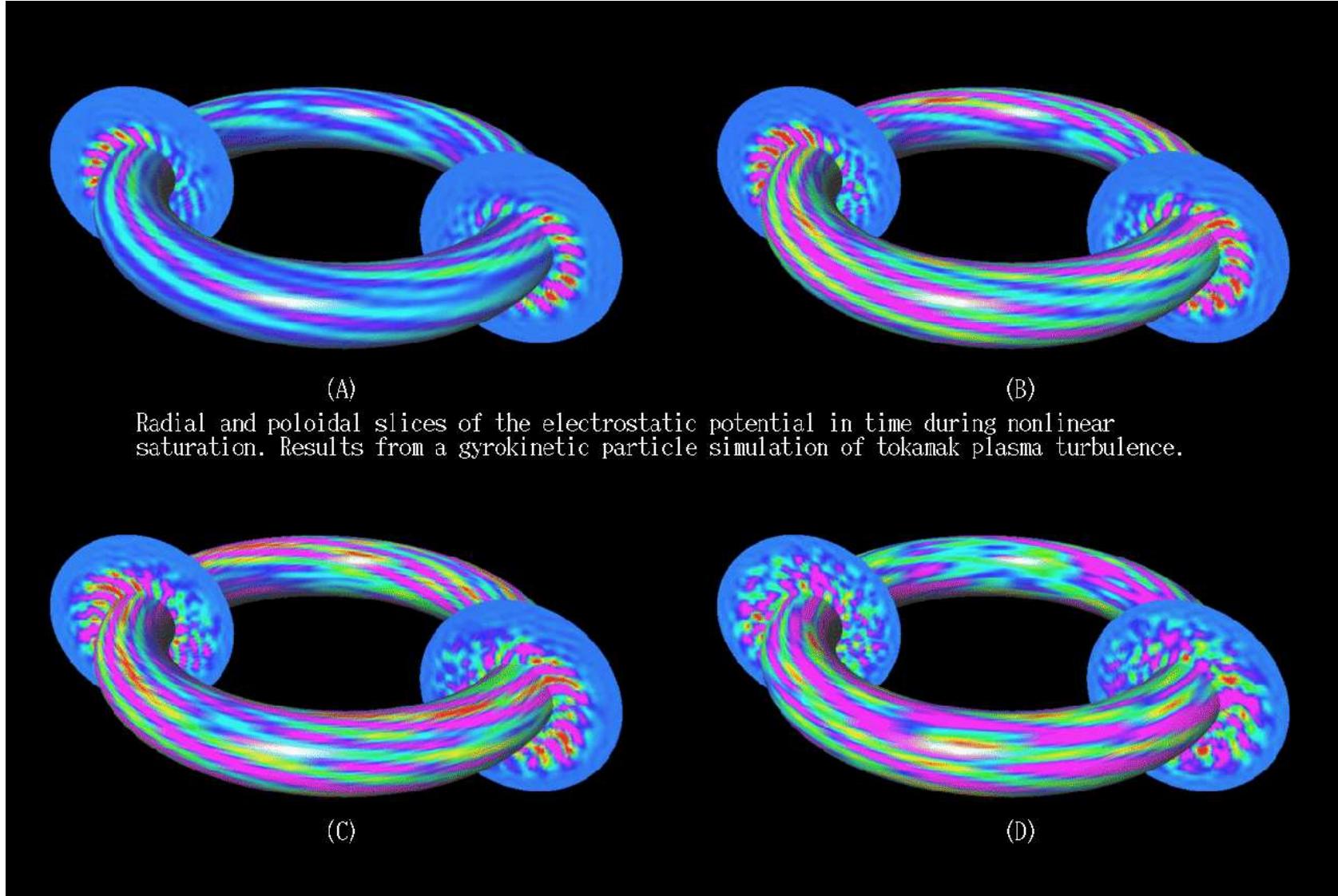
Développement d'ordinateurs pour :

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



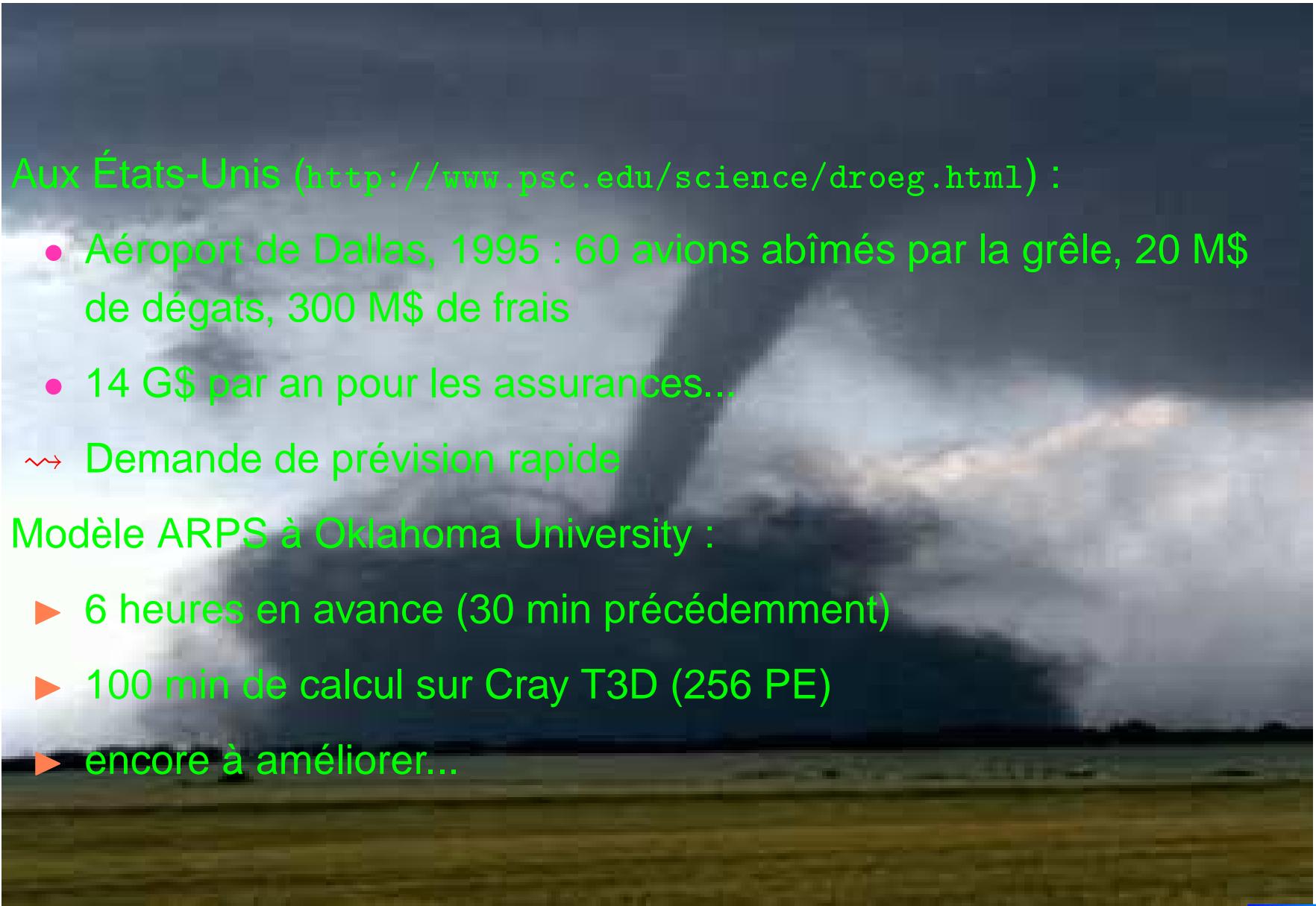
- Simulation et calcul numérique, expérimentation numérique
 - ▶ Environnement
 - ▶ Biologie moléculaire
 - ▶ Conception industrielle
 - ▶ Chimie : 1 ns de simulation demande plus de 300 milliards d'opérations flottantes par seconde (GFLOPS)
 - ▶ Simulations militaires (moratoire nucléaire, ASCI Accelerated Strategic Computing Initiative aux US, 1 G\$)
- Grosses bases de données & *data-mining*, serveur WWW jeux olympiques
- Simulations financières





- Contraintes de temps
- Contraintes de qualité
 - ▶ Prévision à long terme
 - ▶ Résolution spatiale
- Système instable
- Couplage avec océans, etc.
- Besoin de plus de 200 GFLOPS





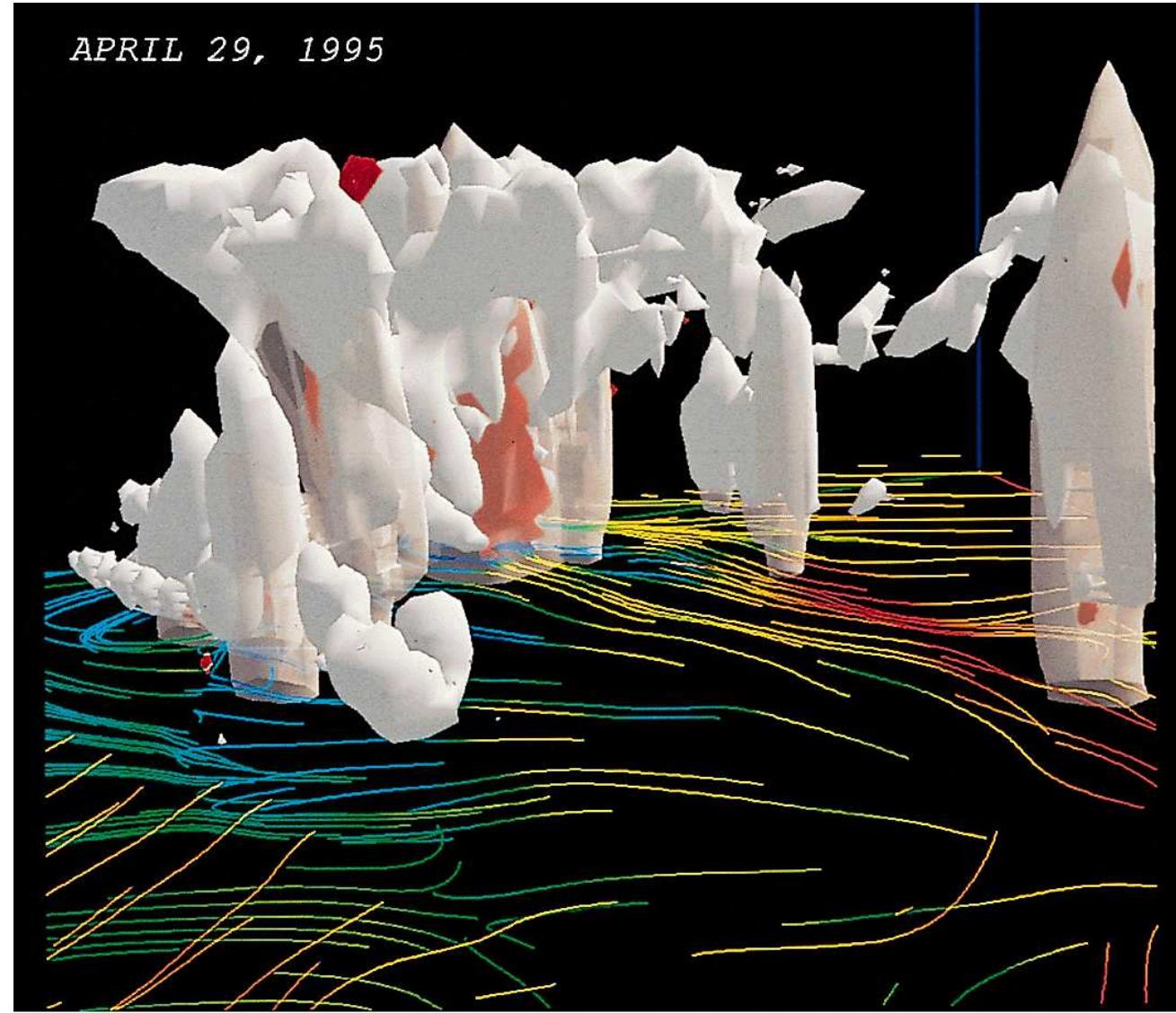
Aux États-Unis (<http://www.psc.edu/science/droeg.html>) :

- Aéroport de Dallas, 1995 : 60 avions abîmés par la grêle, 20 M\$ de dégâts, 300 M\$ de frais
- 14 G\$ par an pour les assurances...
 - ~~> Demande de prévision rapide

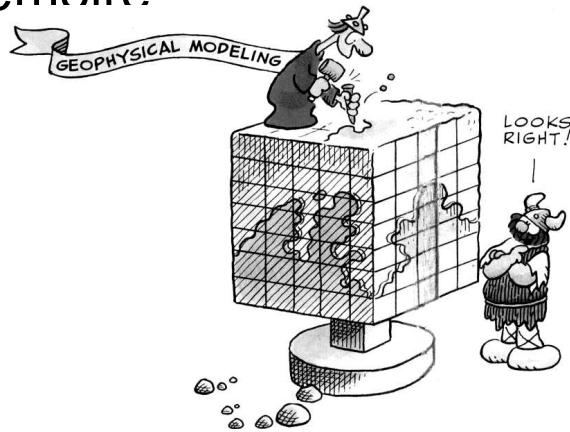
Modèle ARPS à Oklahoma University :

- ▶ 6 heures en avance (30 min précédemment)
- ▶ 100 min de calcul sur Cray T3D (256 PE)
- ▶ encore à améliorer...





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



- Plus de calculs à faire, ↗
vitesse de calcul
 - Débit : beaucoup de calculs aboutissent/unité de temps
 - Latence : temps d'exécution d'une tâche

- Applications limitées par les performances

- Désirs : toujours plus !

$\mu\eta\delta\eta\nu\alpha\gamma\alpha\nu$

- Mesures par programmes étalons (*Benchmark*)

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

supercalculateurs

Gros gains aussi sur les algorithmes...



- *Gordon Bell Awards* délivré lors de la conférence SuperComputing <http://www.sc-conference.org>
- 3 catégories :
 - ▶ Innovation
 - ▶ Plus haute performance soutenue sur une application
 - ▶ Meilleur rapport performance/prix
- 2002 :

Earth Simulator (ES) system is installed in the simulator building (50m x 65m, floor space) at Yokohama Institute for Earth Sciences (Yokohama, Kanagawa) of JAMSTEC. Using NEC SX technology, the ES is the world's fastest supercomputer. It is configured with 640 nodes (64Gflop/s per node, 5,120 CPUs in total), each of which consists of eight vector processors (8Gflop/s per CPU), and achieves 40Teraflop/s peak performance.



To reiterate, the Gordon Bell Award for highest application sustained performance was given to scientists who ran a simulation of a complex climate system using a global atmospheric circulation model on the Earth Simulator, which achieved 26.58 Teraflop/s sustained performance out of the available 40Teraflop/s peak. The Award for Language (special category) was given to scientists who ran IMPACT-3D, an application written in High Performance Fortran (HPF), that simulates the instability in an imploding fluid system for Fusion Science on the Earth Simulator, achieving 14.9 Teraflop/s sustained performance.

These two Gordon Bell Awards were given to eighteen scientists and computer engineers from the Earth Simulator Centre, Japan Marine Science and Technology Centre, NEC Corporation, NEC Informatec Systems, National Institute of Advanced Industrial



Science and Technology, National Space Development Agency and the Atomic Energy Institute of Japan.

The third Gordon Bell Award, for special accomplishment, was given to scientists from the Japan Atomic Energy Research Institute, Earth Simulator Centre, Japan Marine Science and Technology Centre and Nagoya University, who developed new methods for handling the extremely data-intensive calculation of a 3-D Fast Fourier Transform on the Earth Simulator, which allowed researchers to overcome a major hurdle for high performance simulations, in the field of turbulence. This application achieved 16.4Teraflop/s sustained performance on the Earth Simulator.

Two other additional papers earned Gordon Bell Awards for special accomplishment.

The fourth award was given to scientists in the USA, for the



application Salinas, a structural and solid mechanics simulation engineering package of over 100,000 Fortran lines long, that has run on a number of advanced systems, including the ASCI White 12Teraflop/s peak performance system, which achieved 1.16Teraflop/s sustained performance.

The fifth Award was given to scientists from the University of Illinois at Urbana-Champaign, where researchers achieved unprecedented scaling of NAMD, a code that renders an atom-by-atom blueprint of large bio-molecules and bio-molecular systems using thousands of processors.

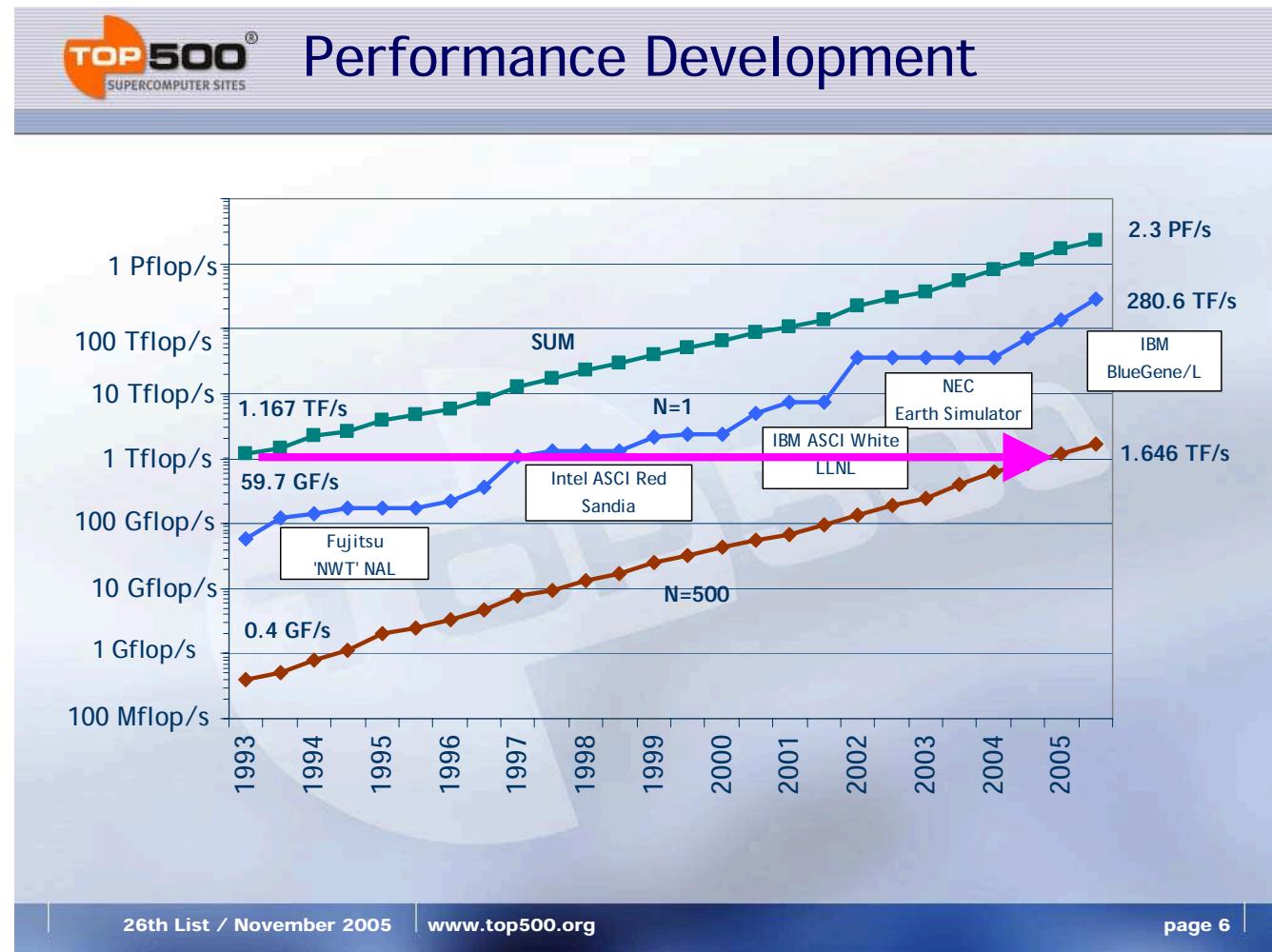
To put it in perspective, the Salinas application achieved 1.16 Teraflop/s sustained performance when run on the 12Teraflop/s ASCI White system. This amounts to between 4.4achieved by the winning applications run on the Earth Simulator.



<http://top500.org>

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





- IBM BlueGene/L au Lawrence Livermore National Laboratory du Département américain de l'énergie (DOE) : culmine à 280 TFLOPS ($2,8 \cdot 10^{14}$ opérations flottantes par seconde) avec 131 072 processeurs
- IBM ASCI Purple dans même laboratoire et construit à base de systèmes *pSeries 575* : 63 TFLOPS avec 10 240 processeurs
- SGI Columbia de la NASA/Ames : 51 TFLOPS
- 2 ordinateurs des Sandia National Laboratories encore du DOE, une grappe à base de PowerEdge de Dell et un Cray XT3 à base d'Opteron ;
- Japonais Earth Simulator de NEC, longtemps première place : relégué à la 7^{ème} place avec ses « modestes » 35 TFLOPS
- Cray XT3 au Oak Ridge National Laboratory du DOE est 10^{ème} avec 20 TFLOPS



Prédominance stratégique des USA... ☹



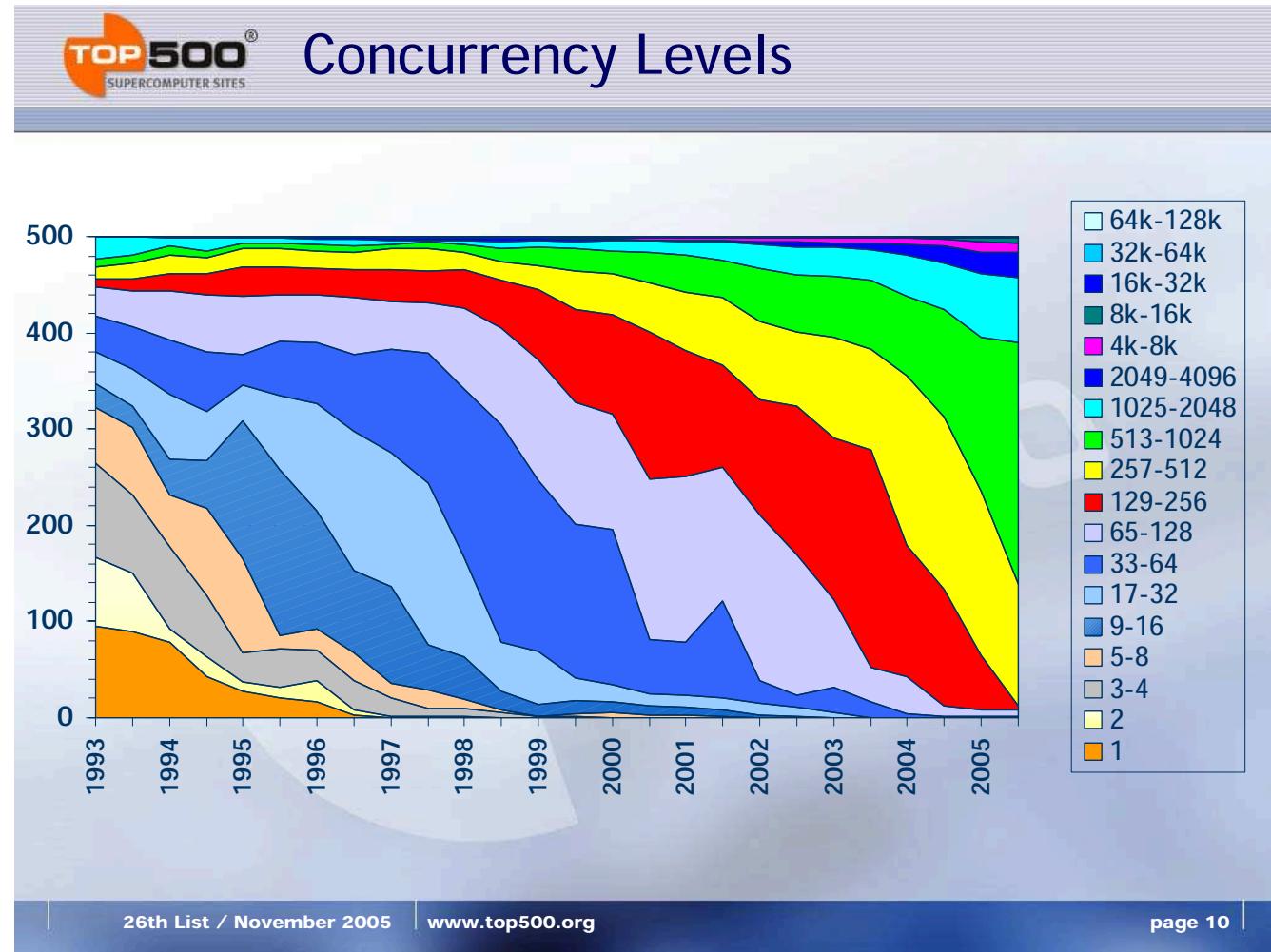
IAHP

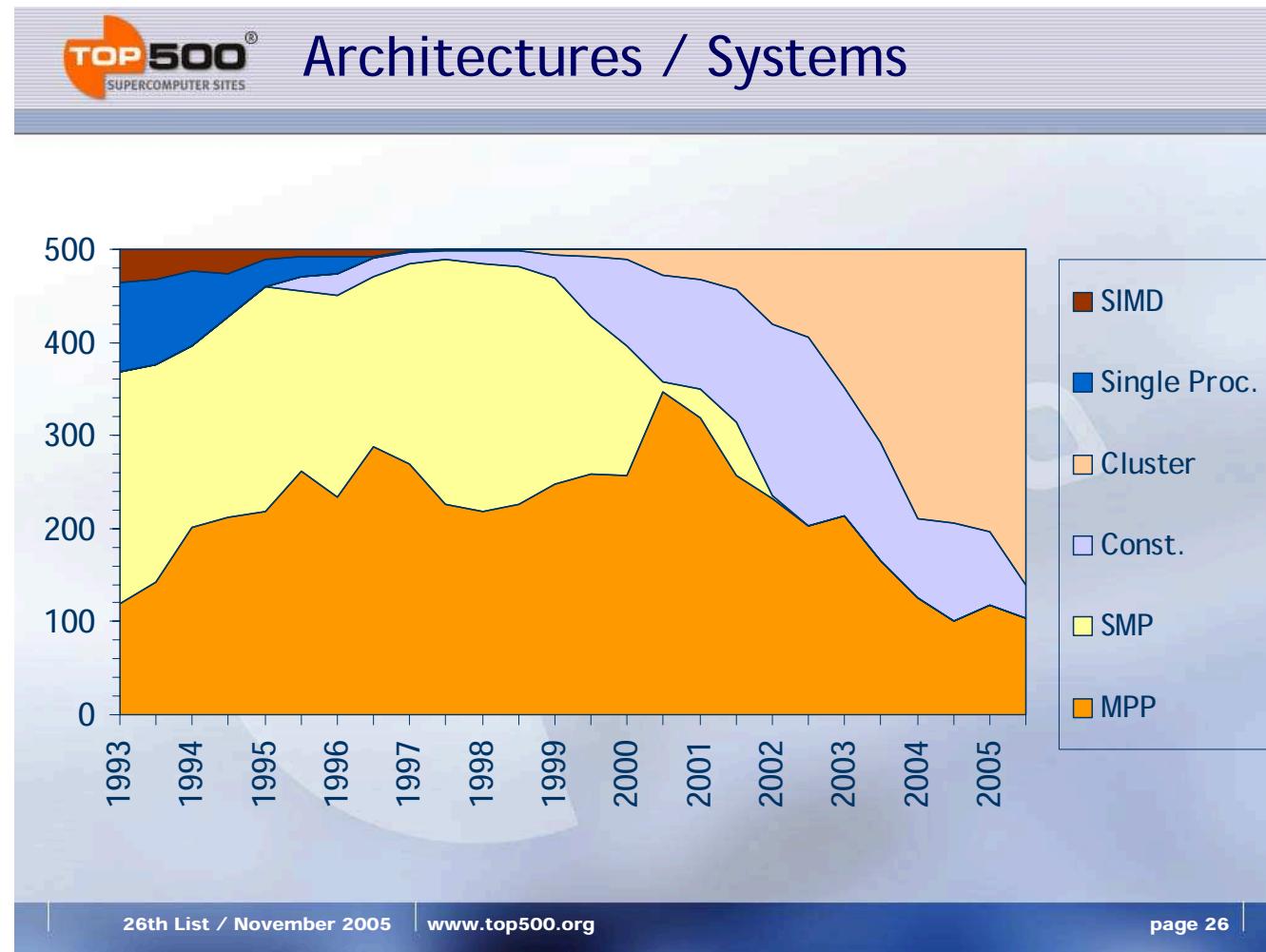
Département Informatique – ENST Bretagne

Architectures Parallèles

• **Introduction**
► **Top 500**

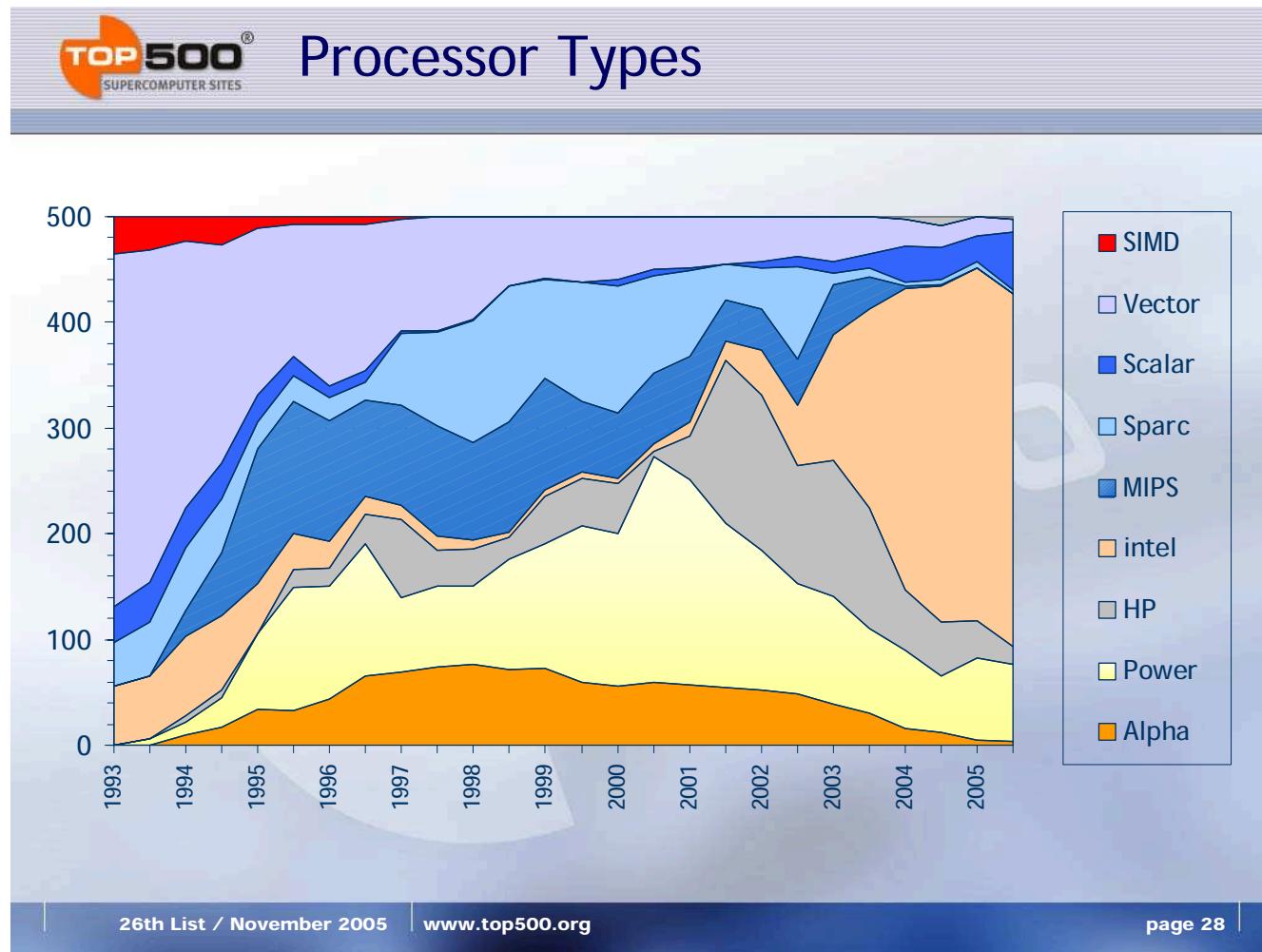






- Tendance à utilisation massive de processeurs standards
- Moins de processeurs vectoriels





IAHP

Département Informatique – ENST Bretagne

Architectures Parallèles

• Introduction
► Top 500



Composantes complémentaires

- Puissance brute des processeurs
- Débit et latence mémoire
- Débit et latence du réseau d'interconnexion
- Entrées-sorties



Dépend de l'application visée ☺



<http://www.top500.org>

- 131 systèmes à plus de 1 TFLOPS
- 57 % des systèmes sont installés aux US
- 90 % des systèmes sont produits aux US
- Top 10
 1. Earth Simulator, 35,86 TFLOPS, Japon
 2. ASCI Q, 13,86 TFLOPS, HP AlphaServer SC, USA DoE Los Alamos National Laboratory
 3. Virginia Tech's X Cluster Institute, 10,28 TFLOPS, Apple G5 + réseau Mellanox/Infiniband, USA
 4. Tungsten cluster, 9,82 TFLOPS, Pentium4 Xeon + Myrinet, NCSA USA
 5. Itanium-2 + Quadrics, DoE PNNL USA



6. Opteron Linux Networx, LANL USA
 7. 2 IBM SP, DoE LVNL et NERSC USA
 8. Pentium4 Xeon, 6,6 TFLOPS, LLNL USA
- Problèmes :
 - ▶ Peu de machines européennes et en Europe ☹
 - ▶ Encore moins de machines française ou situées en France ☹
 - <http://www.top500.org/0RSC/2003> Très intéressant survol des architectures à haute performance



- Histoire ↗ tendances
- Parallélisme intraprocesseur
 - Travail à la chaîne (**pipeline**)
 - Superscalaire & VLIW, **Parallélisme** d'instructions
- Parallélisme inter-processeur & extra-processeur
 - Multiprocesseurs
 - Mémoires rapides
 - Réseaux
 - RAID
- Quelques machines & processeurs
- Programmation & TP...

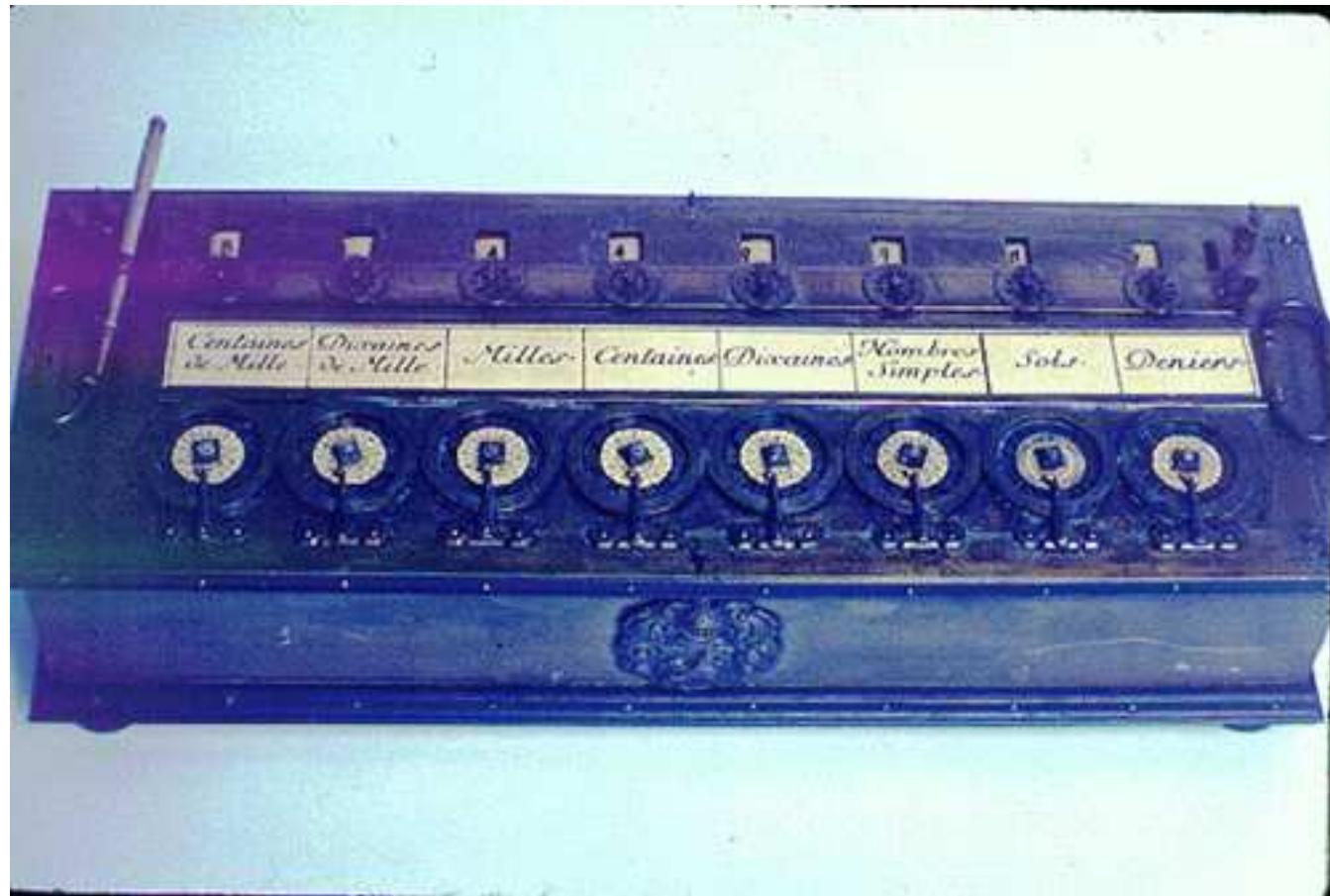


<http://www.computer.org/50/history>

1612 : John NAPIER, première trace des décimaux (nombres flottants...), logarithmes et méthodes de multiplication

1641 : Blaise PASCAL, *pascaline* : additions à 8 chiffres et avec propagation des retenues





1676 : Gottfried LEIBNIZ rajoute la multiplication entière

1703 : Gottfried LEIBNIZ « *Au lieu de la progression de dix en dix, j'ai employé depuis plusieurs années la progression la plus simple de toutes, qui va de deux en deux* » : binaire. Simple car 2 chiffres (bit) ou 2 états. $8_{10} = 1000_2$

1801 : Joseph-Marie JACQUARD, métier à tisser à cartes perforées

1890 : Herman HOLLERITH, contrat pour concevoir des tabulatrices (tri) pour le recensement américain



1925 : Vannevar BUSH, analyseur différentiel pour calcul scientifique (intégration & différentiation)

1935 : Konrad ZUSE, Z-1 calculateur à relais

1936-1939 : John Vincent ATANASOFF and John BERRY, ABC pour résoudre des systèmes linéaires en physique. Mémoire régénérative, unité arithmétique et logique, cartes perforées

1939 : George STIBITZ, Bell Labs Model 1 avec nombres complexes. Commande à distance avec « télétype » par téléphone... Internet ?

1944 : Équipe de Tommy FLOWERS & Alan TURING, Colossus Mark I pour décrypter les messages allemands, 2400 tubes

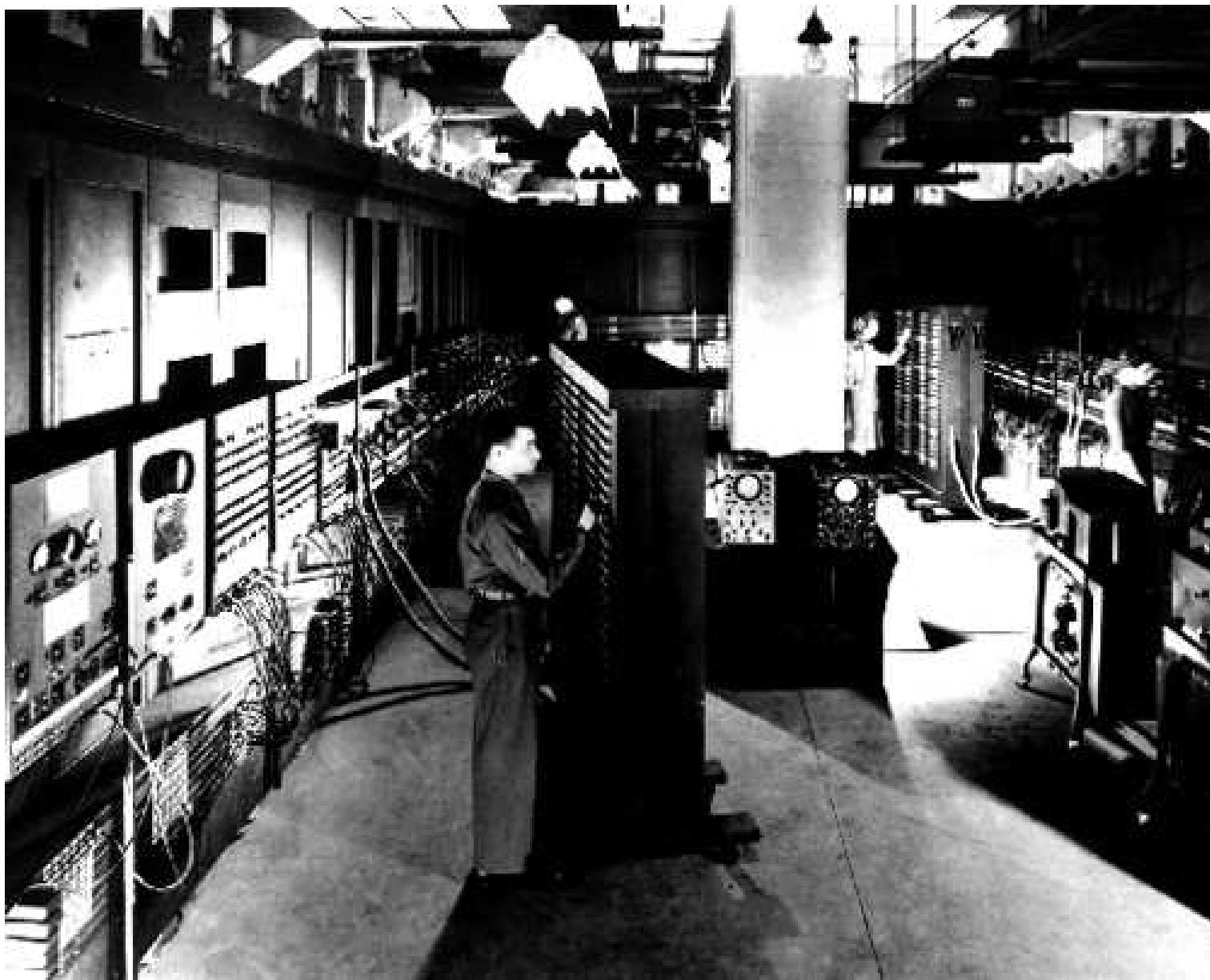


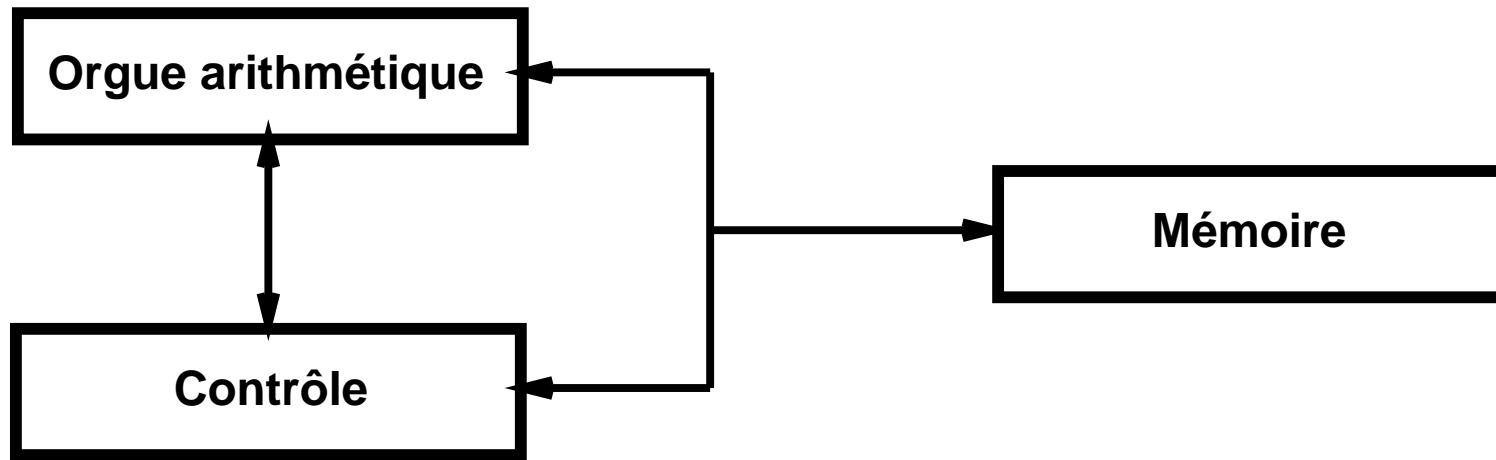


Architecture des calculateurs numériques généraux séquentiels :
Principes de John VON NEUMANN (EDVAC, 1945), Wallace ECKERT & Presper MAUCHLY

1946 : Wallace ECKERT & Presper MAUCHLY, ENIAC. 18000 tubes à vide, 30 tonnes, 174 kW, 5000 +/s, 333 ×/s à 10 chiffres, modifié plus tard avec programme en mémoire







1948 : Équipe de Max NEWMAN, prototype du Manchester Mark I, 1^{er} ordinateur à programme en mémoire

1951 : Jay FORRESTER & Bob EVERETT, Whirlwind, simulateur temps réel à mémoire à tores, 500000 +/s, 50000 ×/s, \$32 000 de tubes/mois

¿ Comment augmenter les performances ?



- Orgue arithmétique plus rapide ou plusieurs orgues
- Unité de contrôle plus sophistiquée ou plusieurs
- Mémoire plus rapide ou plusieurs
- Bus plus rapide ou plusieurs

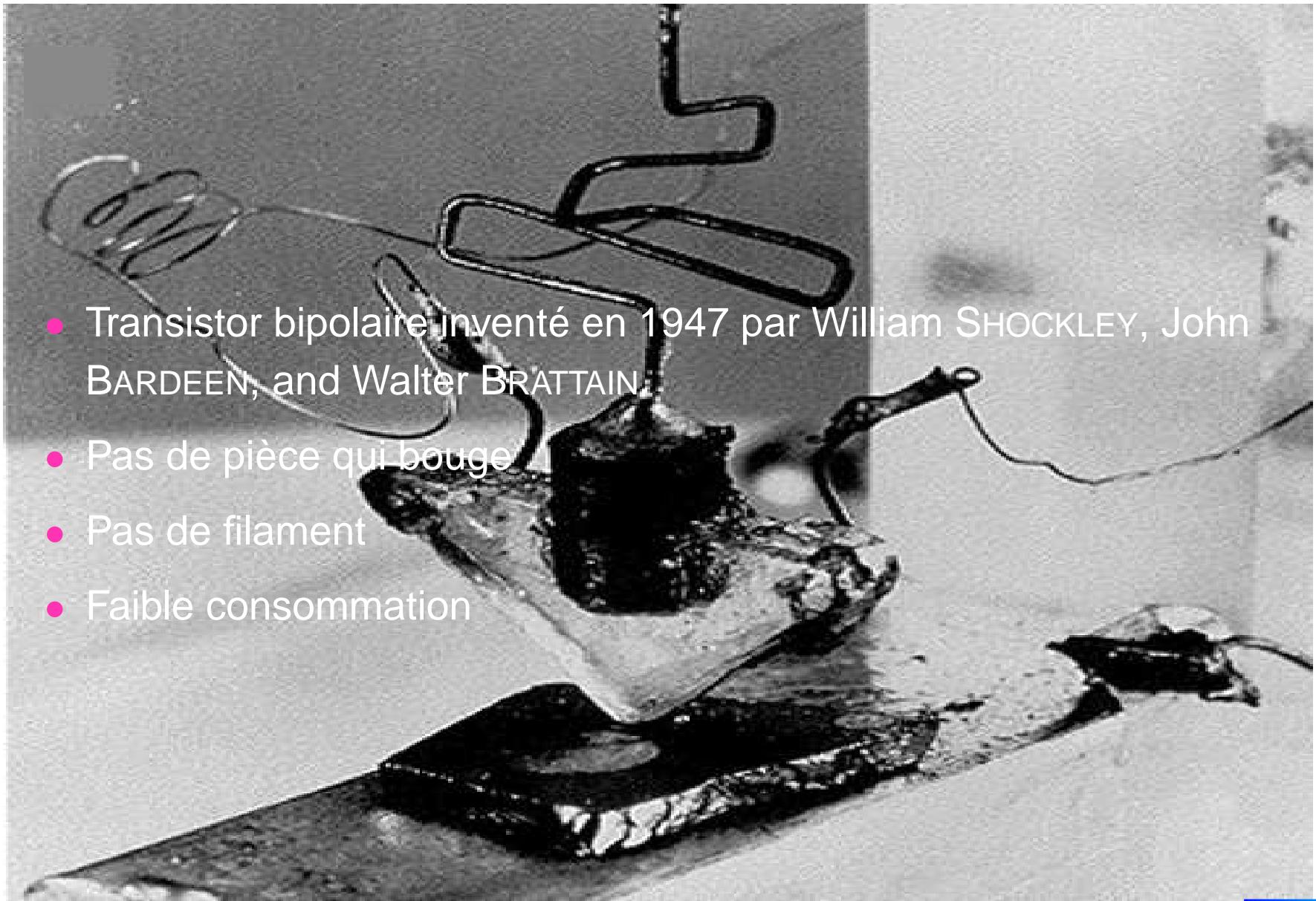
Toutes les combinaisons possibles et « ou » inclusifs...

¡ Vaste espace de choix !

[Amdahl, 1967] : la partie la plus lente *pour une application donnée* domine...

~~> machine bien équilibrée *par application*





1954 : IBM 704, nombres flottants, 5 kFLOPS

1957 : John BACKUS (IBM), langage & compilateur Fortran ↗
vitesse de programmation

1957 : Disque dur IBM 305 RAMAC (\approx 2 réfrigérateurs)

1957 : Seymour CRAY, CDC1604, supercalculateur tout transistor

1958 : Circuit intégré

1959 : Bull Gamma 60, instruction de parallélisme (car mémoire trop rapide)



1961 : IBM 7090/94, supercalculateur, temps partagé

1964 : Seymour CRAY, CDC6600, plusieurs unités fonctionnelles, RISC, mémoire démultipliée en bancs, 10 MIPS (Millions d'Instructions Par Seconde) [Thornton, 1964]

Généralisation pour la vitesse d'utilisation :

- Langages de haut niveau plus proches des algorithmes & compilateurs
- Systèmes d'exploitation : logiciels d'aide au fonctionnement (démarrage des programmes, entrées-sorties,...)



1967 : Début des ordinateurs à base de circuits intégrés

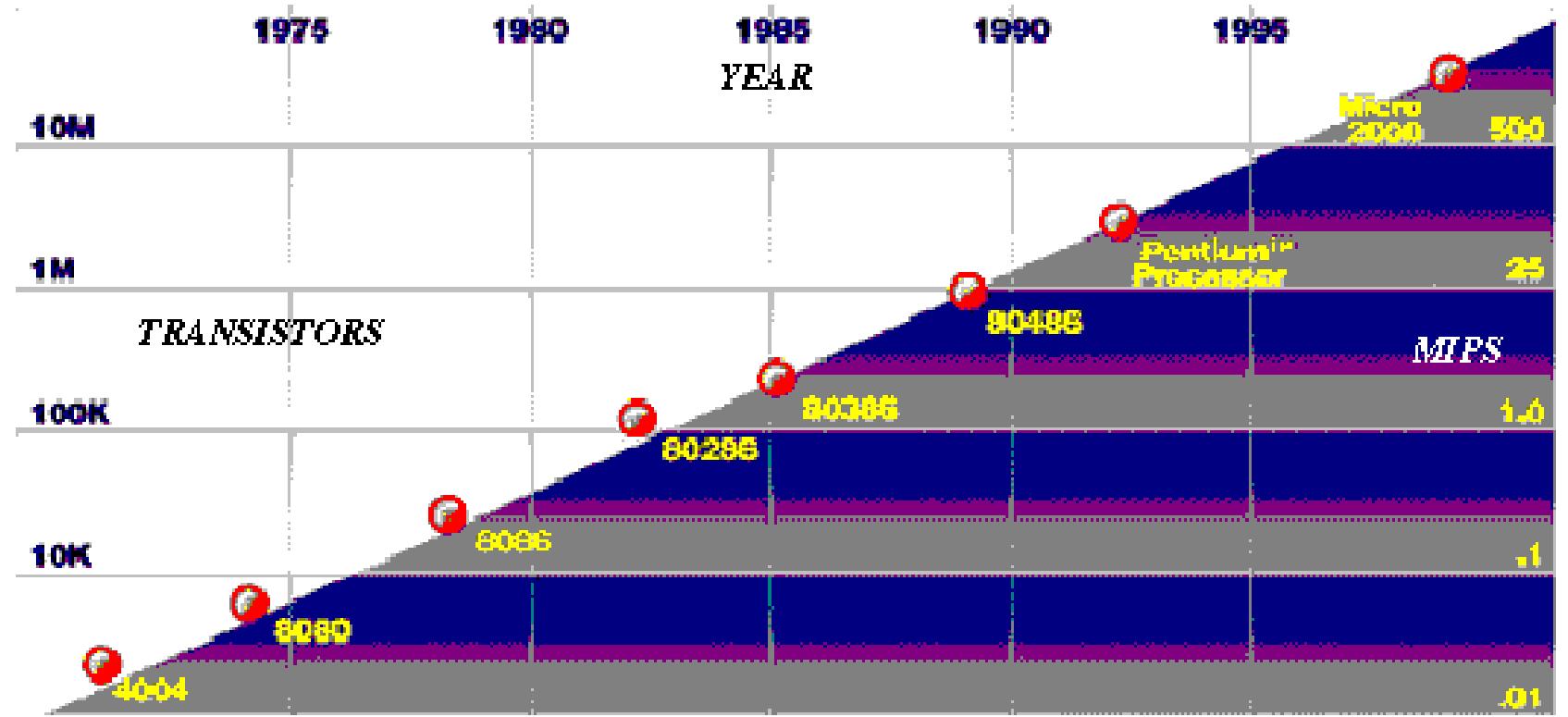
1967 : IBM360/91, exécution spéculative, dans le désordre producteur-consommateur

1971 : Ted HOFF, Intel 4004, microprocesseur pour calculette

1976 : Cray 1, supercalculateur **vectoriel, RISC, registres vectoriels**, liaisons courtes ($1\text{ ns} \approx 30\text{ cm}$), 250 **MFLOPS** (millions d'opérations flottantes par seconde) par chaînage d'unités vectorielles, mémoire rapide (1 mot/cycle), ECL ↗ refroidissement par fréon



- Rapide ↗ petit ↗ ordinateur intégré dans 1 circuit
- Loi de Gordon MOORE sur la progression du nombre de transistors ($\times 4$ tous les 3 ans)



- Performances ↗
- Marché de masse ↝ ↓ prix
- Intégrations des techniques classiques des supercalculateurs ↝ briques de base des supercalculateurs
- Problème d'entrées-sorties et de débit mémoire car informations transitent par les « pattes », en nombre limité (500)



1985 : INMOS Transputer T414, processeur « Lego », données 32 bits

1986 : Cray X-MP, 4 processeurs, 713 MFLOPS sur LINPACK 1000.
Règle 1 Mo pour 1 MFLOPS

1986 : CM-1, 65536 PEs (processeurs élémentaires) 1 bit, **SIMD**, 4 Ko/PE

1991 : CM-5, 1024 coprocesseurs flottants, 256 processeurs 32 bits, **MIMD**

1996 : Mort de Seymour CRAY

1996 : Cray T932, **MIMD vectoriel** 32 processeurs de 1,8 GFLOPS, 57,6 GFLOPS, mémoire SRAM rapide globale (chère)

1996 : Cray T3E-900, **MIMD** 2048 processeurs à 450 MHz, 1,8 TFLOPS

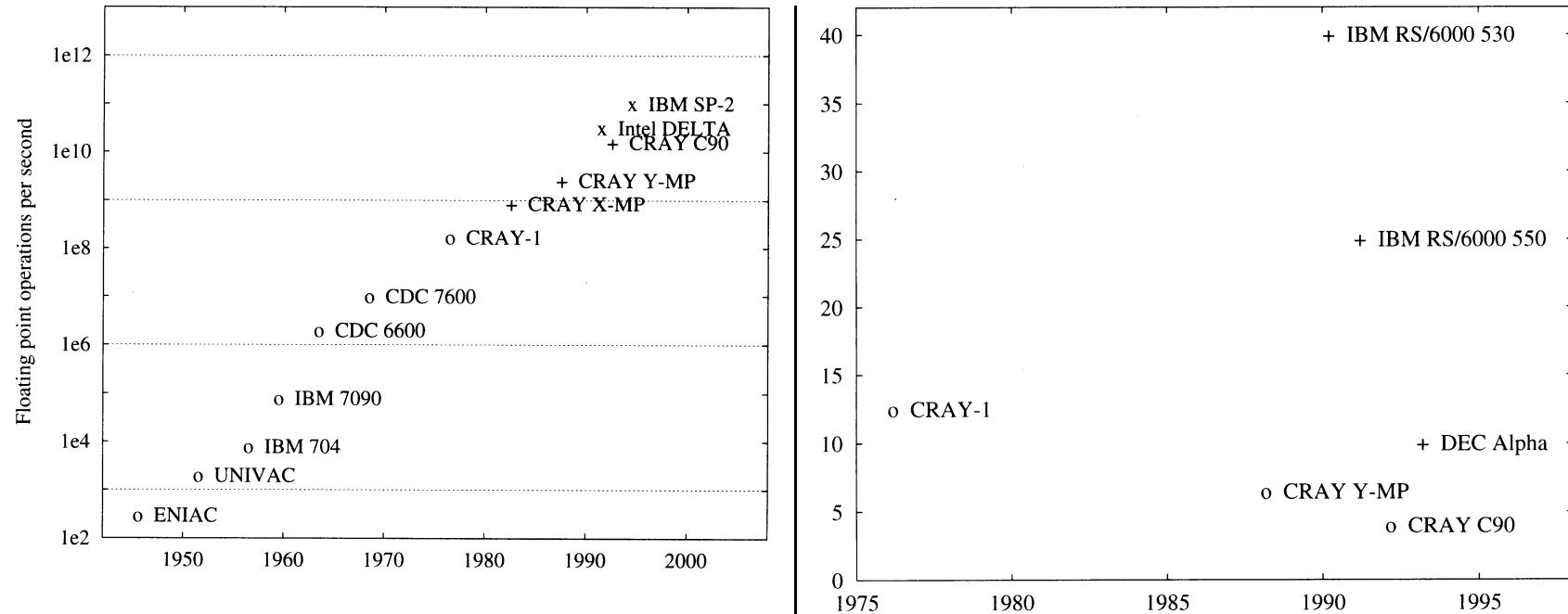
1996 : Disque dur IBM 2,5", 1,6 Go, 99 grammes



1997 : Intel ASCI red & IBM ASCI blue : 1+ TFLOPS

Beaucoup de compagnies ont fait faillite... Petit marché





Temps de cycle (ns)



Domaine tiré par la technologie...

Extrapolation de l'informatique à la voiture sur 100 ans :

- $\frac{1}{2}$ dé à coudre d'essence aux 1000 km
- 5 000 000 km/h
- 0,1 €

Adaptation d'impédance
[Hennessy et al., 1990] :

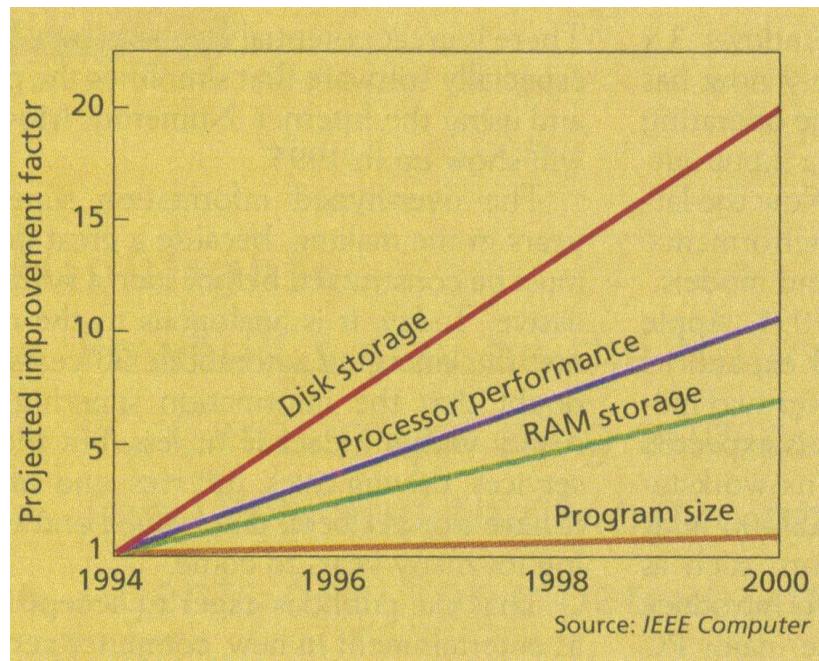
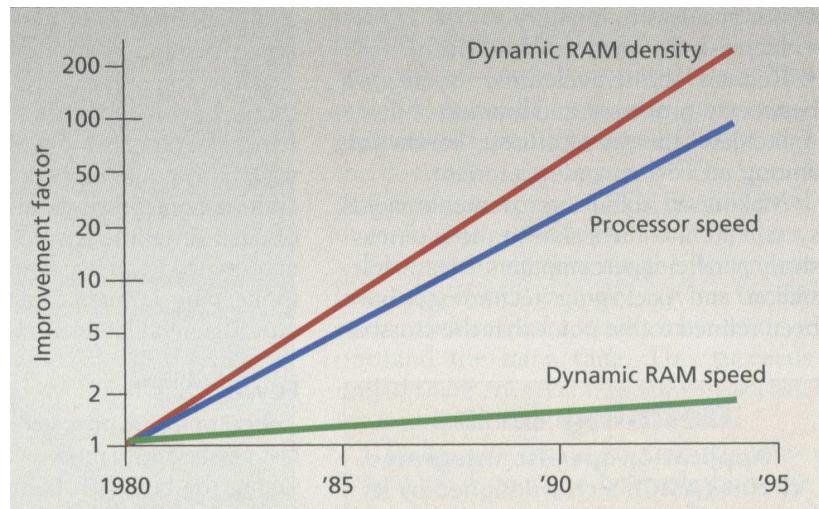
- Technologie : nombre de transistors $\nearrow +18$ à $+35\%/\text{an}$

- Processeurs : $\nearrow +60\%/\text{an}$ ($\times 4$ en 3 ans)
- Toujours plus \rightsquigarrow gain architectural supplémentaire

La mémoire :

- Technologie par boîtier $\approx 2\$$ courants quelle que soit l'époque
- Évolution $\nearrow +60\%/\text{an}$ ($\times 4$ en 3 ans)
- Besoins de $\nearrow +50$ à $+100\%/\text{an}$ (+0,5 à 1 bit/an)





Disques :

- ▶ ↗ +25–+60 %/an, 2,35 €/Go en 2002m 1 €/Go en 2004
- ~~> non suffisant

Supercalculateurs :

- ▶ ∀ époque, prix ≈ 10 M\$ courants. Ordinateurs les plus performants et les plus... chers
- ▶ nécessité de baisser encore plus les coûts/performance

! Il faut faire quelque chose !



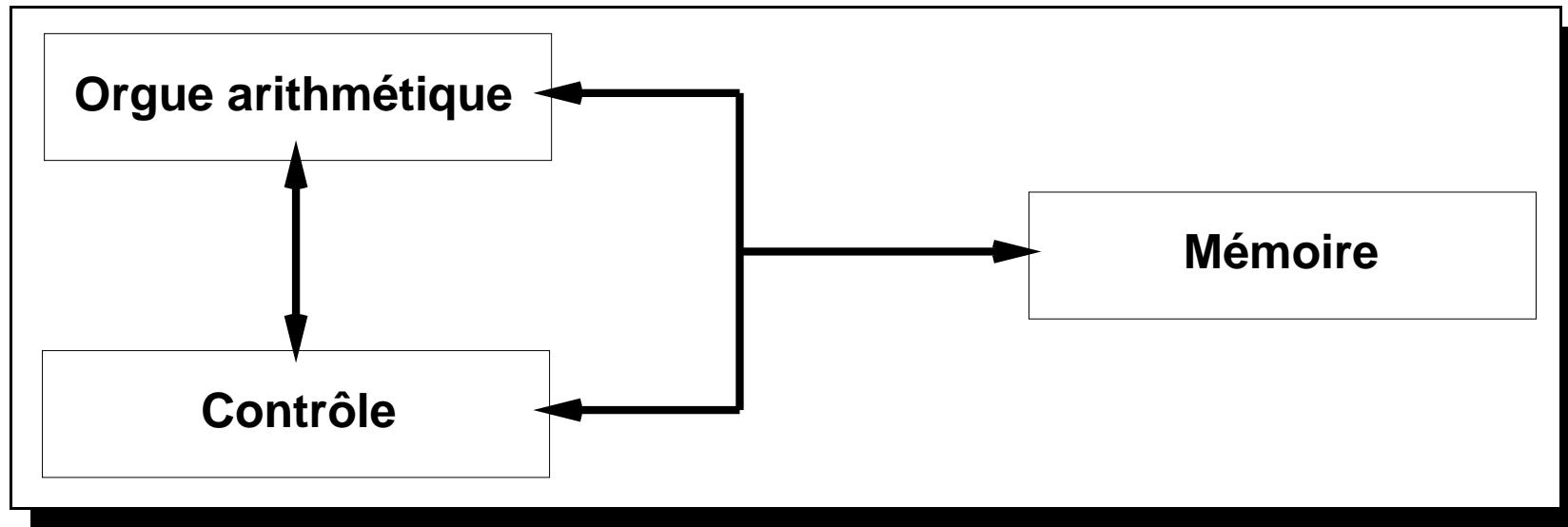
- Temps de réponse : voulu par l'utilisateur !
≈ Synonyme : temps d'exécution, latence.
- Débit ou bande passante : voulus par le directeur d'un centre de calcul.

Méthodes duales pour augmenter les *performances* = $\frac{\text{débit}}{\text{latence}}$:

- ↘ Latence
- ↗ Débit



Principes de ECKERT, MAUCHLY et von NEUMANN, années 1940 : une mémoire contient des données *ET* un programme
~~> concept d'ordinateur universel



+ Entrées-sorties sur un des chemins de données.

¿ Limitations ?



- Problèmes de fiabilité :
 - Composants très petits : kT , rayonnement ionisant, HEISENBERG, ...
<http://www.ewh.ieee.org/r6/scv/rs/articles/ser-050323-talk-ref.pdf>
How Cosmic Rays Cause Computer Downtime, Ray Heald,
IEEE Rel. Soc. SCV Meeting, 23/3/2005
 - Redondance des composants = sorte de //isme démocratique
- Vitesse :
 - Vitesse de propagation : petite machine
 - Petite machine et composants rapides qui dissipent : évacuer la chaleur ☺
- Coût :
 - Composants rapides



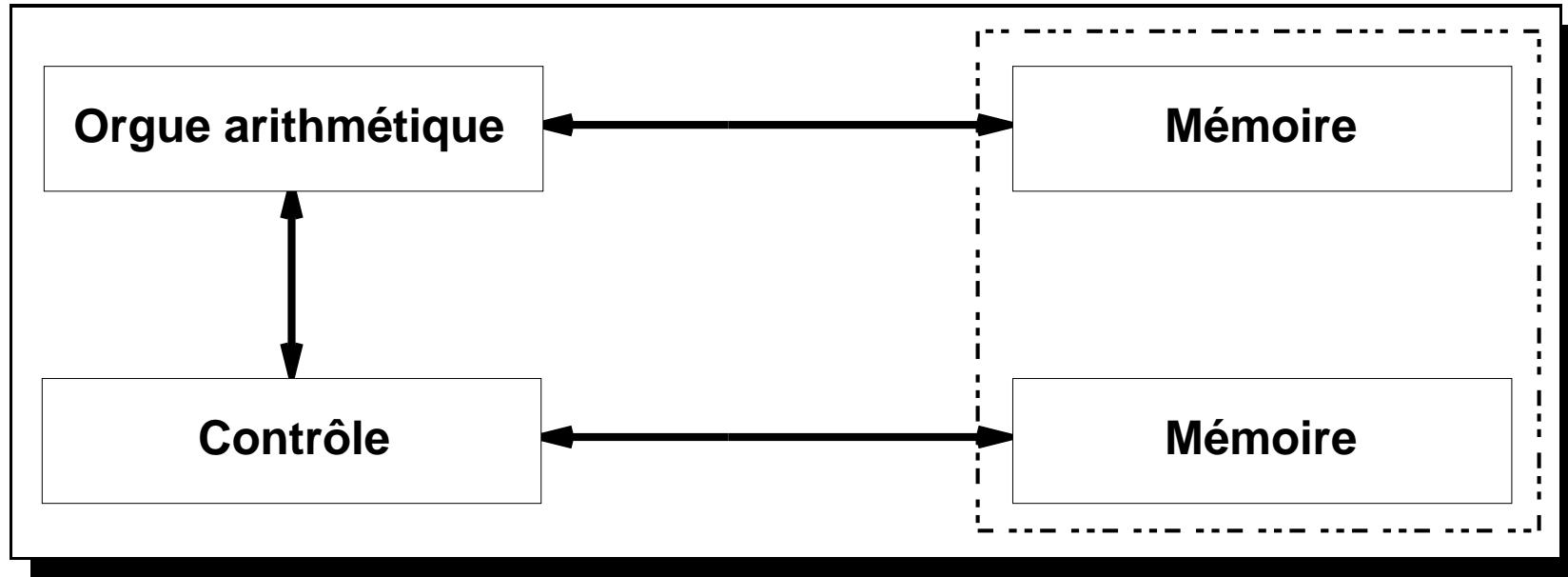
- Réfrigération
- Ne fait qu'une chose l'une après l'autre

~~~ //isme.

(PS : ce discours existe depuis +50 ans...)



Parallélisme de base : répliquer les chemins de données



- ▶ Chemin de données pour le programme
- ▶ Chemin de données pour les données (*sic*)
- ... Chemin de données pour les entrées-sorties



Pourquoi les ordinateurs ne sont pas tous « parallèles » ?

Implique développement harmonieux du parallélisme :

- matériel
- modèles de programmation
- langages parallèles
- compilation
- algorithmique adaptée
- compatibilité et portabilité logicielles
- utilisateur raisonne séquentiellement sur des concepts...
- Pas que des applications high-tech...

¡ Depuis 50 ans le matériel attend le logiciel !

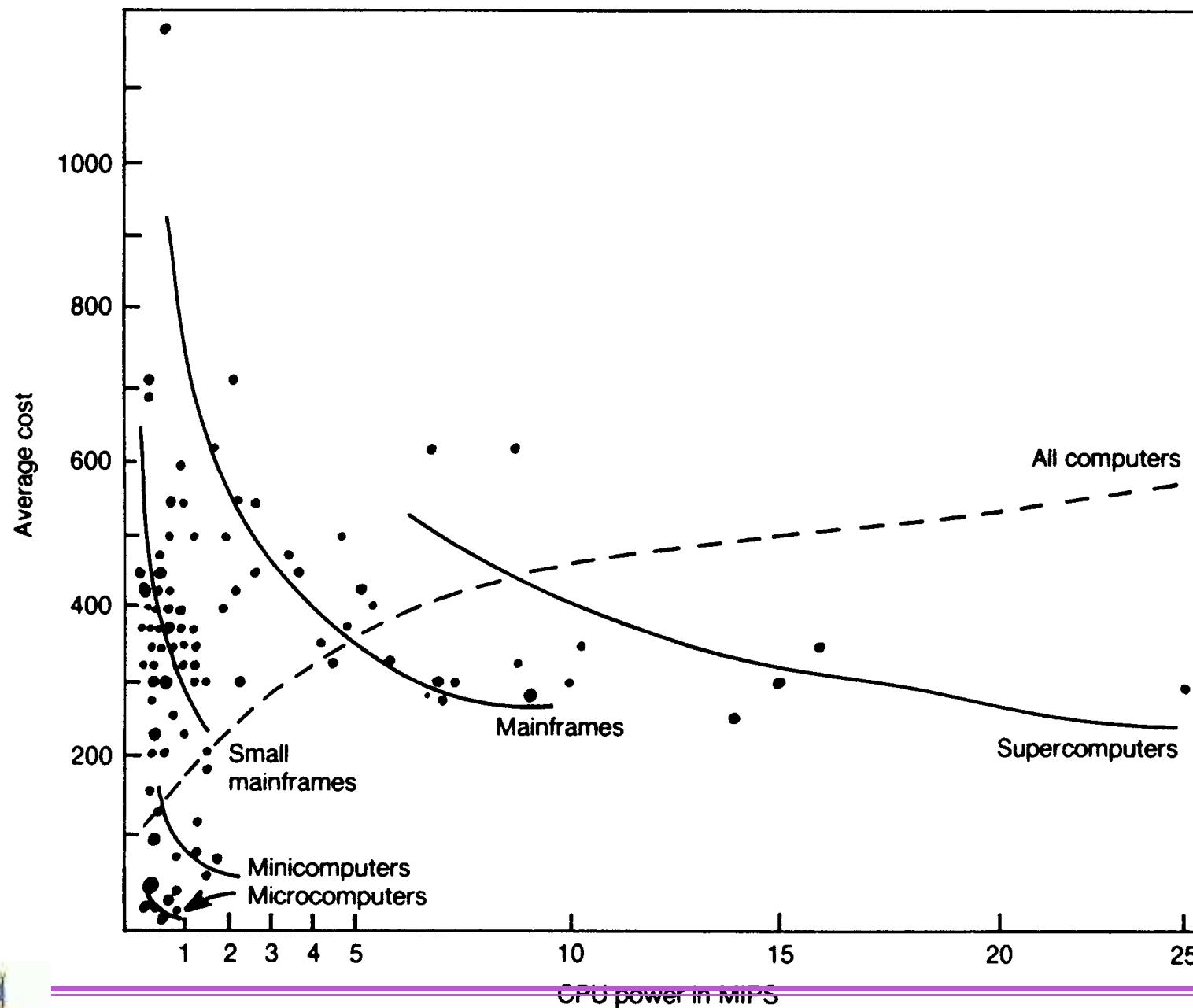


Informatique = interaction subtile d'économie et de technologie

Des années 1940 à 1975 : « loi de Grosch »

$$\frac{\text{coût}}{\text{performance}} \sim \frac{1}{\sqrt{\text{performance}}}$$





... à classe identique !



Revisite entre les classes [Ein-Dor, 1985] :

$$\frac{\text{coût}}{\text{performance}} \begin{pmatrix} \text{supercalculateur} \\ \text{mainframe} \\ \text{minimainframe} \\ \text{miniordinateur} \\ \text{microordinateur} \end{pmatrix} \approx \begin{pmatrix} 1474.4 \\ 856.8 \\ 303.9 \\ 98.5 \\ 6.2 \end{pmatrix} \text{ puissance}^{-0.55}$$

~~ de la marge pour :

- Faible efficacité des machines parallèles
- Coût du réseau
- ...



Trouver un compromis entre matériel et logiciel pour diminuer les coûts à performance constante.

1. Optimiser le cas courant  $\rightsquigarrow$  RISC (analyse quantitative)
2. Chiffrer les optimisations (utilisation de la loi d'AMDAHL).
3. Souvent principe de localité spatiale et temporelle.
4. Prévoir l'évolution des performances de la technologie !

Le logiciel est parfois plus rapide que le matériel !



Cf. RISC et CISC. Syndrome de l'*usine à gaz*...



<http://toyvax.glendale.ca.us/~vance/vaxbar.html>



« Accélération » (speedup) :

$$a = \frac{T_A}{T_B}$$

Avec 1 et  $N$  processeurs :

$$a_N = \frac{T_1}{T_N}$$

Et accélération absolue :

$$a_N = \frac{S_{\text{meilleur}}}{T_N}$$

Efficacité ou linéarité :

$$\ell_N = e_N = \frac{a_N}{N}$$

Approximation du coût du calcul

$$c_N = N \times T_N$$



## Hypothèse 1 :

Programme = partie séquentielle + partie parallélisable

$$\begin{cases} T_1 = s + p \\ T_N = S + P \end{cases}$$

## Hypothèse 2 : normalisation $T_1 = 1$ et

$$\begin{cases} S = s \\ P = \frac{p}{N} \end{cases}$$

Alors [Amdahl, 1967] :

$$\begin{cases} a = \frac{1}{s + \frac{p}{N}} \\ \ell = \frac{1}{Ns + p} \end{cases}$$

=: On ne peut pas dépasser  $a_\infty = \frac{1}{s}$  !



~~> Optimiser la partie coûteuse



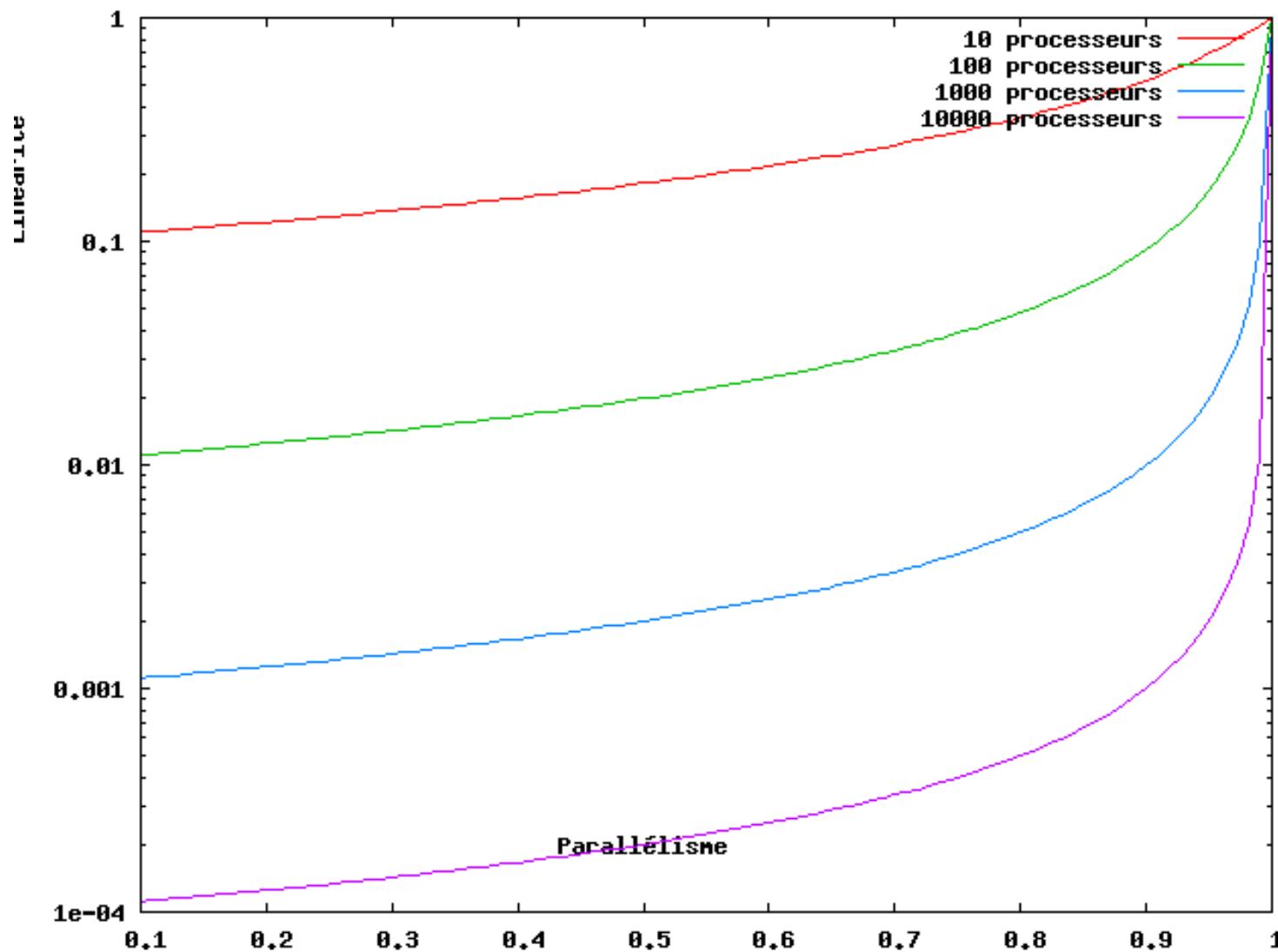
$$s = 5\% =: a_\infty = 20$$

$$a_\infty = 1000 =: s < 0,1\%$$

Le parallélisme ne sert-il vraiment à rien ?

1. Supposition : pas de recouvrement parallèle/séquentiel [Flynn, 1972]. Important pour  $N$  moyens !
2. Modèle fruste : pas de parallélisation de  $s$  [Kuck, 1976]
3. Suppose aucune évolution dans les algorithmes
4. Approche non psychologique.





Superlinéarité impossible [Faber et al., 1986, Faber et al., 1987] :

1. soit  $\mathcal{A}$  meilleur algorithme séquentiel pour résoudre notre problème
2. soit un algorithme superlinéaire  $\mathcal{B}$  pour machine à  $N$  processeurs :  $T_N^{\mathcal{B}} < \frac{T^{\mathcal{A}}}{N}$
3. simuler la machine parallèle avec un monoprocesseur  $\rightsquigarrow \mathcal{C}$  séquentiel
4.  $T^{\mathcal{C}} = NT_N^{\mathcal{B}} < T^{\mathcal{A}}$  : contradiction

Problème : les considérations matérielles sont *ignorées* !

Pas de changement de contextes, contentions structurelles (registres, caches, mémoire,...).



Superlinéarité possible : plus de caches et de registres dans la machine parallèle que dans la machine séquentielle, factorisation de matériel [Kim et al., 1991, :keryell] et de manière plus théorique mémoire parallèle PRAM [Akl et al., 1992]



- Communications interprocesseur
  - ▶ Échanges d'information
  - ▶ Synchronisation
- Réseaux d'interconnexion
  - ▶ Bande passante finie
  - ▶ Latence non nulle

=: souvent linéarité  $\ell \leq 1$ ,  
parfois  $\ell \ll 1$ .



Solutions éventuelles :

- Diminuer les communications
- Localiser les communications
- Recouvrir communications et calculs
- Accélérer matériellement les communications
- Logiciels plus sophistiqués

(Ralentir les processeurs n'est pas une bonne solution... ☺)



Depuis 1878 [Gustafson, 1988] :

- Temps de réponse constants (des humains aux commandes ☺)
- ↗ Taille et complexité des problèmes

La partie séquentielle n'évolue pas :

**Problème constant** : accélération d'un problème donné  
[Amdahl, 1967]

**Temps constant** : traiter un problème plus gros dans le même temps [Gustafson, 1988]

Si on néglige les problèmes de communication :

| Plus de limite !



$$\left. \begin{array}{l} T_1 = s + p \\ T_N = S + P \\ T_N = 1 \\ s = S \\ p = N \times P \end{array} \right\} =: \left\{ \begin{array}{l} a = S + NP = 1 + (N - 1)P \\ \ell = \frac{a}{N} = 1 + \frac{1 - P}{N} \end{array} \right.$$

Le gain de puissance de calcul sert à traiter des problèmes plus gros dans le même temps



Tests :

- Programmes réels (compilateur gcc, T<sub>E</sub>X, Spice, *votre code*) interpellant les utilisateurs
- Noyaux extraits de programmes (Linpack, Livermore)
- Jouets (crible d'Érastosthène, puzzle, quicksort)
- Benchmarks synthétiques (Whetstone (flottant), Dhrystone (entier)) : créés pour représenter des comportements de programmes réels.
- MIPS = Millions d'Instructions Par Secondes ↗ GIPS
- MFLOPS = Millions d'Opérations Flottantes Par Secondes ↗ GFLOPS, TFLOPS, PFLOPS, EFLOPS, ...



Les MIPS et MFLOPS sont à utiliser dans un contexte donné



## Problèmes :

- Importance du compilateur (version, niveaux d'optimisations, triche)  
<http://www.spec.org/cpu2000/results/res2003q4/cpu2000-20031117-02620.as>  
253.perlbmk: -DSPEC\_CPU2000\_LP64 -DSPEC\_CPU2000\_NEED\_BOOL  
                  -DSPEC\_CPU2000\_LINUX\_IA64 -DSPEC\_CPU2000\_GLIBC22
- Importance du système (version, charge, graphisme, single user, réseau,...)
- Comportements différents sur les tests selon les machines

Performances très différentes de la performance crête (garantie de non dépassement!).

Perfect Benchmark : en général 1 % de la performance crête



Performance ↵ plusieurs choses à la fois,  
mais où paralléliser?

| ↑ Gros grain              |                                 |
|---------------------------|---------------------------------|
| utilisateur               | multiutilisateur                |
| programme                 | multiprogrammation              |
| tâche                     | multitâche                      |
| fibres ( <i>threads</i> ) | SVR4, Posix, Java,...           |
| instruction               | pipeline, VLIW et superscalaire |
| intra-opération           | mathématiques discrètes         |

↓ Grain fin



Conditions d'indépendance de 2 parties  $i$  et  $j$  (BERNSTEIN, 1966) :

$$R_i \cap W_j = W_i \cap R_j = W_i \cap W_j = \emptyset$$

Types de dépendance de données :

|                 |            |                |            |         |
|-----------------|------------|----------------|------------|---------|
| instruction $i$ | A = ...    | ... = A        | A = ...    | ... = A |
| instruction $j$ | ... = A    | A = ...        | A = ...    | ... = A |
|                 | $\delta^t$ | $\bar{\delta}$ | $\delta^o$ | (ras)   |
|                 | RaW        | WaR            | WaW        | RaR     |
|                 | def-use    | use-def        | def-def    | use-use |

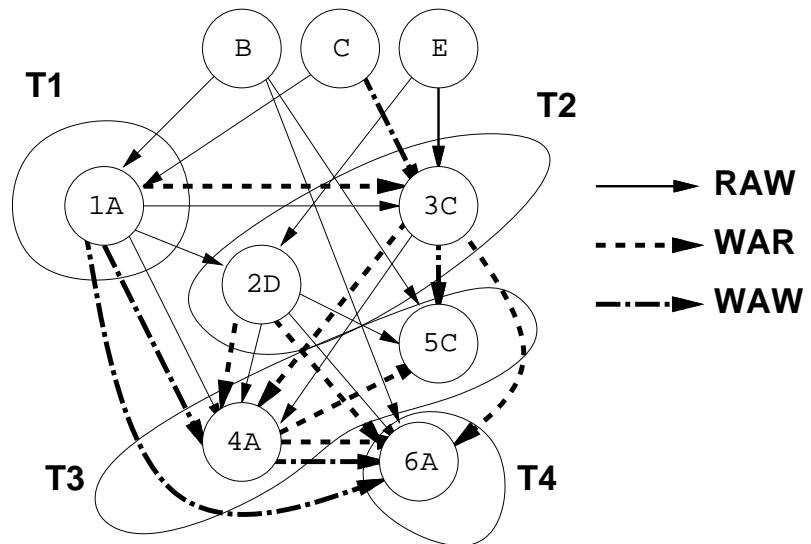
+ Dépendances de contrôle.



Exécuter le plus de choses en parallèles sans enfreindre de dépendance : parcourir au plus tôt le graphe de dépendance topologiquement

- 1:  $A = B + C$
- 2:  $D = A * E$
- 3:  $C = A - E$
- 4:  $A = D - A * C$
- 5:  $C = D - B$
- 6:  $A = B/D$

~~ Parallélisme maximal de 1,5



Problèmes :

- Matériel fini
- Boucles : graphe cyclique
- Dépendances de contrôle (branchements)
- Planification en temps non polynomial
- Bonnes heuristiques...

Compromis entre

- ▶ Parallélisme statique (compilation)
- ▶ Dynamique à la demande (à l'exécution)



La théorie :

- Temps d'exécution
- Coût d'exécution
- Performances
- Consommation électrique, embarqué

La pratique ☺ :

- Pour le plaisir !
- Faire des thèses...
- Faire de la recherche et des articles...
- Faire des cours sur la question ?



- Parallélisme à grain fin
- Très efficace car pas besoin de sortir d'un CI aujourd'hui
- *relativement* transparent pour l'utilisateur

$$\text{Temps}_{\text{CPU}} = N_{\text{instructions}} \times \text{Cycles}_{\text{instruction}} \times \tau_{\text{cycle}}$$

Diminuer temps CPU :

- $\searrow \tau_{\text{cycle}}$  : faire des instructions + simples, RISC
- $\searrow \text{Cycles}_{\text{instruction}}$  et/ou  $\searrow \tau_{\text{cycle}}$  : exécuter à la chaîne les phases d'exécution des instructions
- $\searrow \text{Cycles}_{\text{instruction}}$  : plus d'1 instruction par cycle : superscalaire, VLIW et vectoriel

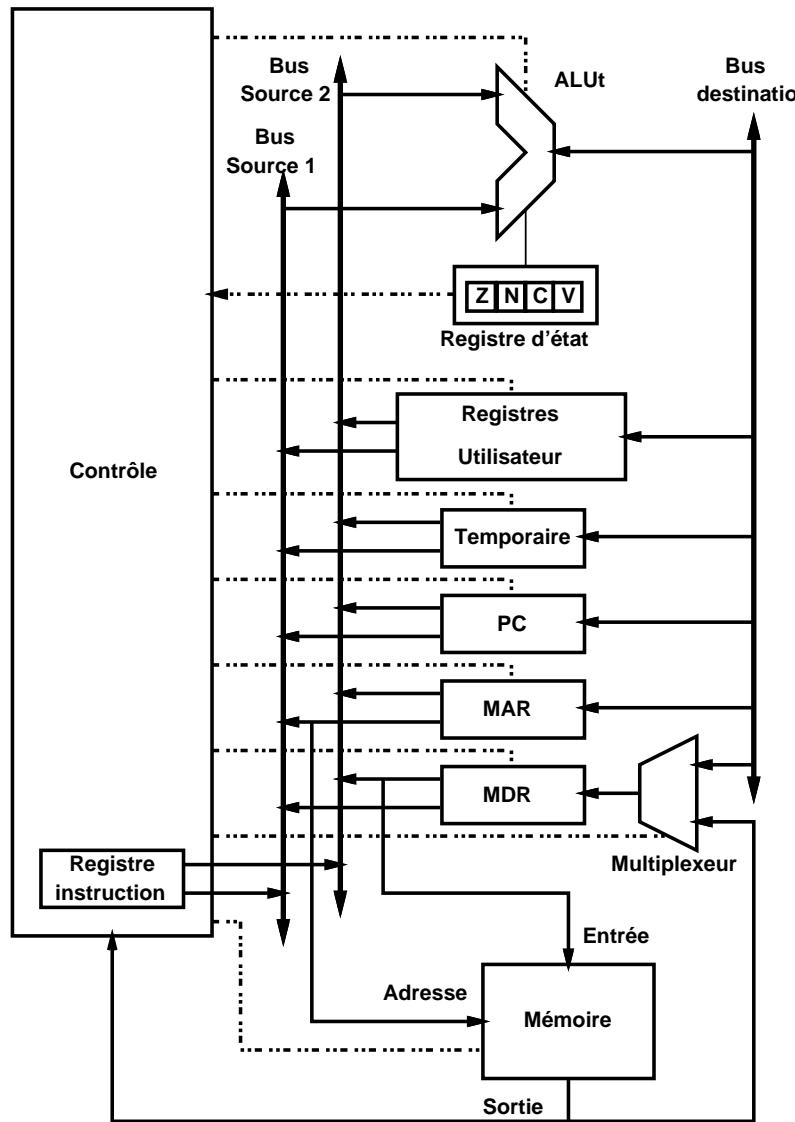


Exécuter des instructions en mémoire (pour simplifier, instructions rangées dans l'ordre en mémoire ☺) :

1. Lire instruction à l'adresse PC (Program Counter)  $\equiv *PC$
2. Décoder instruction
3. Lire opérandes
4. Effectuer opération
5. Ranger résultat
6. Incrémenter PC  $\equiv PC++$

Entrées-sortie : zone mémoire spéciale par exemple

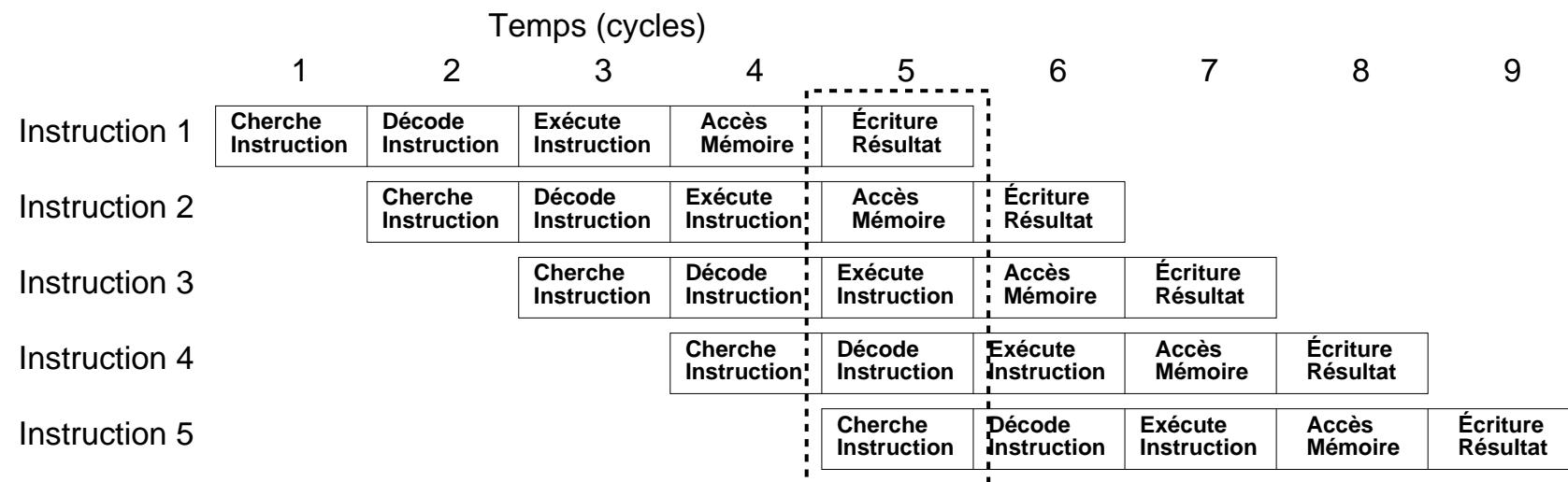




Travail à la chaîne appliqué aux instructions (IBM7030 Stretch) :

| Cherche Instruction | Décode Instruction | Exécute Instruction | Accès Mémoire | Écriture Résultat |
|---------------------|--------------------|---------------------|---------------|-------------------|
|---------------------|--------------------|---------------------|---------------|-------------------|

1. Découpage du travail en  $p$  phases successives : « étages »
2. Isoler temporellement chaque phase
3. Faire progresser de manière synchrone le travail



Avantages :

- 1 résultat fourni à chaque avancement du pipeline
- Gain de  $p$  en première approximation
- Peu de matériel rajouté : 1 verrou/étage (magique !)



- Pipeline synchrone : temps de cycle = phase la plus longue =: équilibrage des étages
- Temps de propagation des verrous : chaque phase est plus longue

$$a = \frac{\bar{T}_{\text{instructions sans pipeline}}}{\bar{T}_{\text{instructions avec pipeline}}}$$

Exemple :  $a = \frac{260}{65} = 4$ .



**Paradoxe** : globalement programme plus rapide mais latence des instructions plus importante !

- ▶ Accélération maximale plafonnée par le temps de traversée des verrous
- ▶ Compromis latence-débit (pipeline—superpipeline)
- ▶ Compromis à trouver sur le nombre d'étages

Accélération maximale ?

$$\text{Solution : AMDahl donne } a_{\infty} = \frac{5}{260} = 52.$$

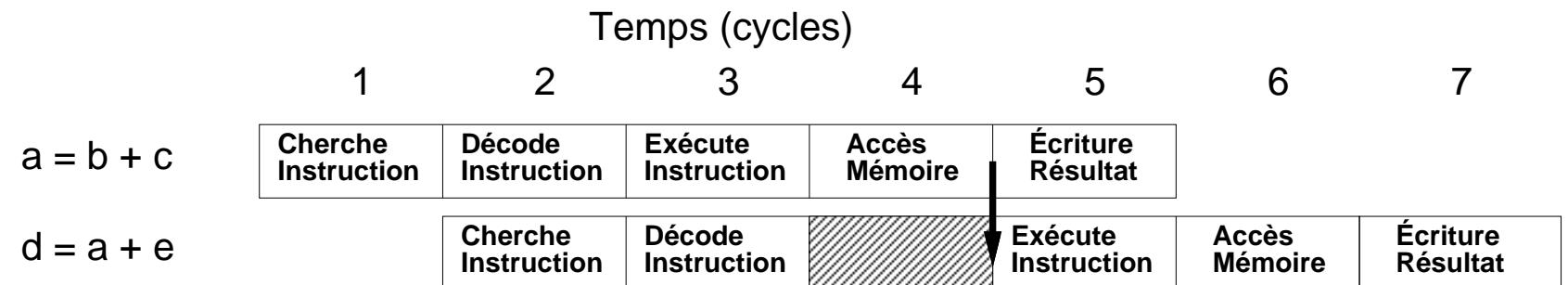


Exemple du DLX (processeur d'école) :

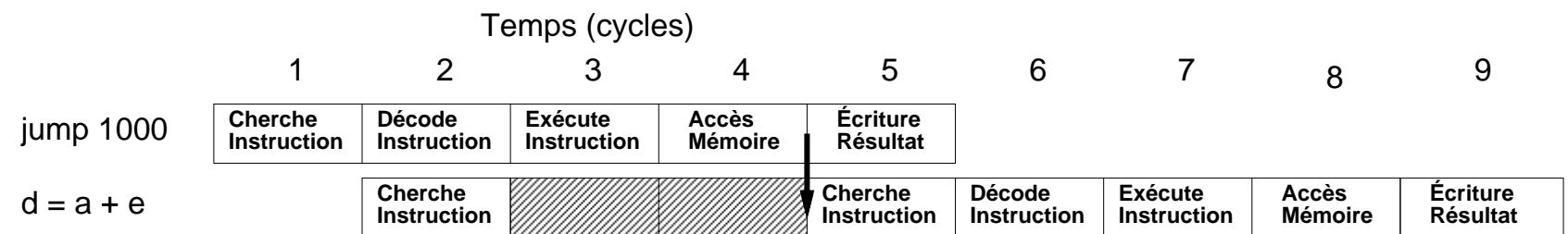
- Nécessite un débit mémoire  $\times 5$
- 1 accès code + 1 accès donnée par cycle  
~~> Architecture HARVARD, ou moins interne



- Accidents structurels :
  - ▶ Conflit sur des ressources matérielles (mémoire, opérateur,...)
- Accidents conjoncturels — respect de la sémantique du programme :
  - ▶ dépendances de données : producteur-consommateur



- ▶ dépendances de contrôle : branchements



=: geler le pipeline pendant  $g$  cycles pour résoudre les problèmes...

↙ vitesse

↗ accidents avec ↗ étages

$$a = \frac{\tau_{\text{cycle non pipeliné}}}{\tau_{\text{cycle pipeliné}}} \times \frac{CPI_{\text{non pipeliné}}}{CPI_{\text{pipeliné}} + g}$$



Problème structurel : supposons que

- 30 % des instructions accèdent à la mémoire
- un seul port d'accès à la mémoire
- $CPI_{\text{non pipeliné}} = 1$
- $CPI_{\text{pipeliné}} = 1.2$  (autres problèmes)
- $\tau_{\text{cycle non pipeliné}} = 260 \text{ ns}$
- $\tau_{\text{cycle pipeliné}} = 65 \text{ ns}$

Exercice : calculer  $a_{\text{idéal}}$  (sans gel dû à la mémoire) et  $a$ .

$$a_{\text{idéal}} = \frac{260}{65} \times \frac{1}{1,2} \approx 3,33$$
$$a = \frac{260}{65} \times \frac{1,2 + 1 \times 0,3}{2,66} \approx 1,25$$
$$= \frac{a_{\text{idéal}}}{a_{\text{idéal}}} = 1,25$$

Mémoire multiport utile ? Fonction du coût...



Vraie dépendance : une instruction a besoin du résultat d'une instruction précédente

Idée : éviter le passage par les registre

*(feed) forwarding*

Si accès mémoire dans l'ordre, pas de problèmes de dépendance au niveau de la mémoire

Si ressource lente, obligé de geler le pipeline.

Exemple : mémoire, coprocesseur flottant



**WaR** : impossible dans le modèle simple

- Lecture au début du pipeline
- Écriture en fin de pipeline

Problème par exemple si autoincrément ADD R2,R1,R3 et LD  
R2,(R1)+

**WaW** : impossible car les écritures sont en séquence et au même endroit (WB)



Compiler

=

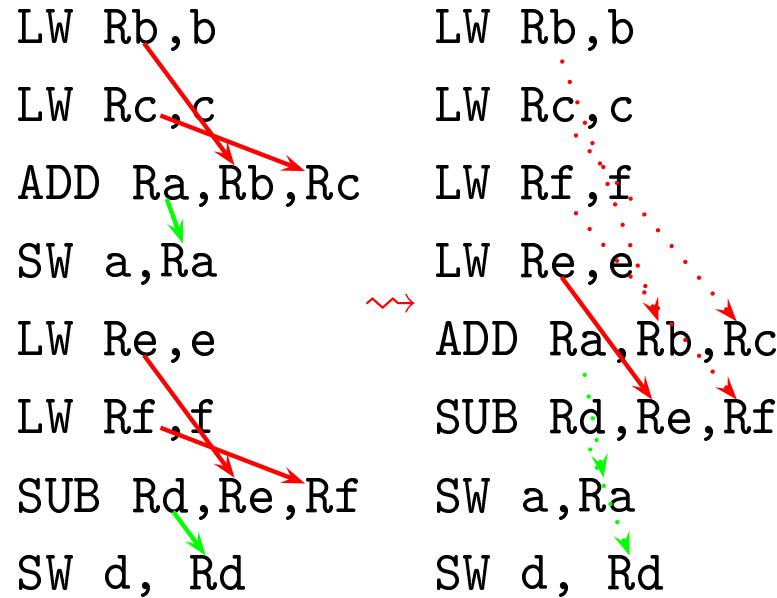
réarranger le code pour  
limiter les gels du pipeline

Exercice : optimiser le code

$$a = b + c$$

$$d = e - f$$





Dépendances : 2 cycles, 3 cycles, en pointillé si non bloquant

Passe de 14 cycles à 9 cycles

Efficace mais nécessite beaucoup de registres... 😞

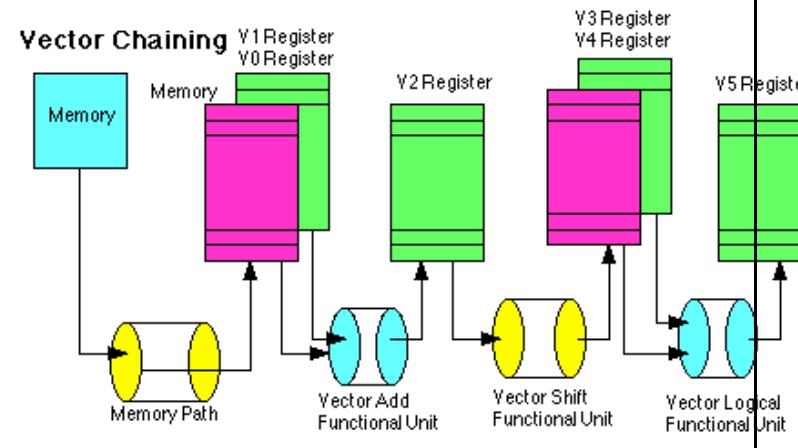


Cray-1 [Russel, 1978], 250

MFLOPS crête

- 1 instruction  $\equiv$  opération vectorielle ( $\approx$  boucle de calculs)

```
do i = 1, 1000
    a(i) = b(i) + c(i)
```

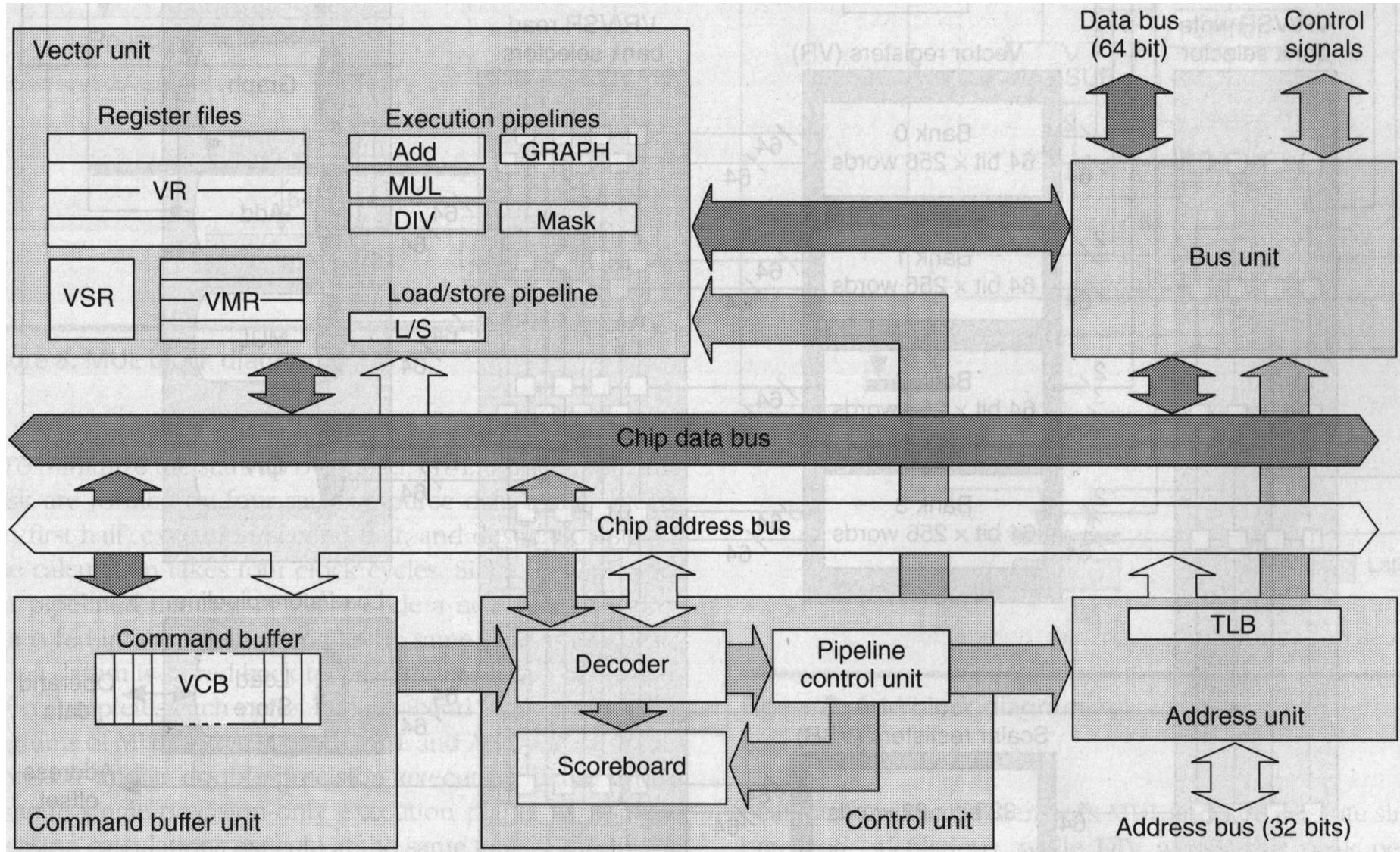


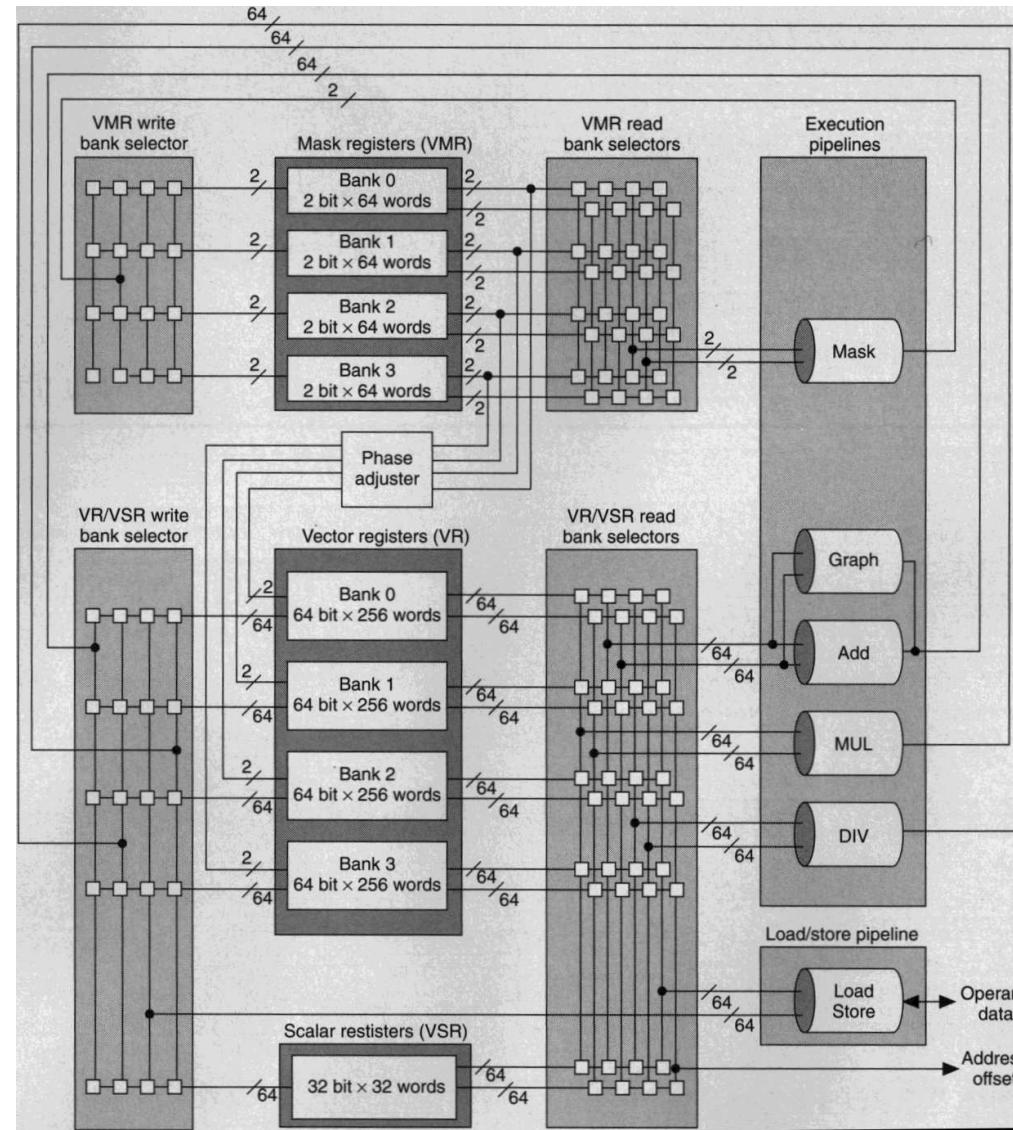
- Extension du pipeline aux opérations vectorielles
- Chaînage des unités vectorielles producteur-consommateur
- Registres vectoriels
- Bon débit mémoire nécessaire
- Code adapté au très long pipeline (amorçage et dépendance),  
« vectorisation » nécessaire

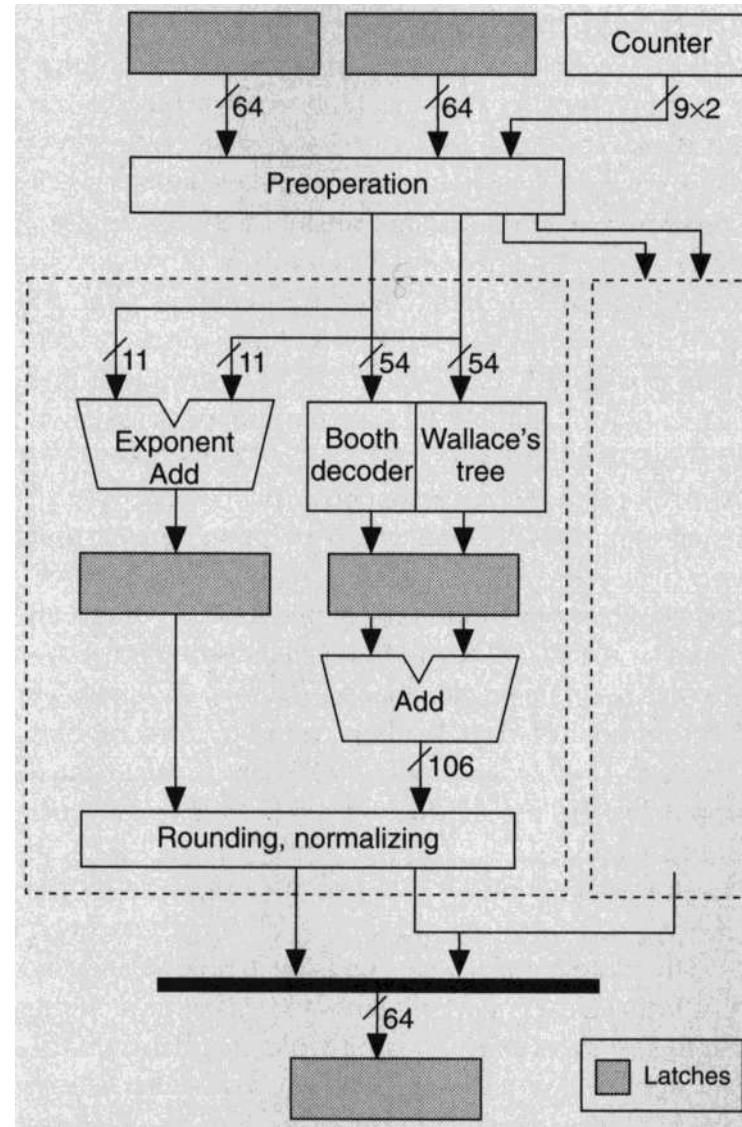


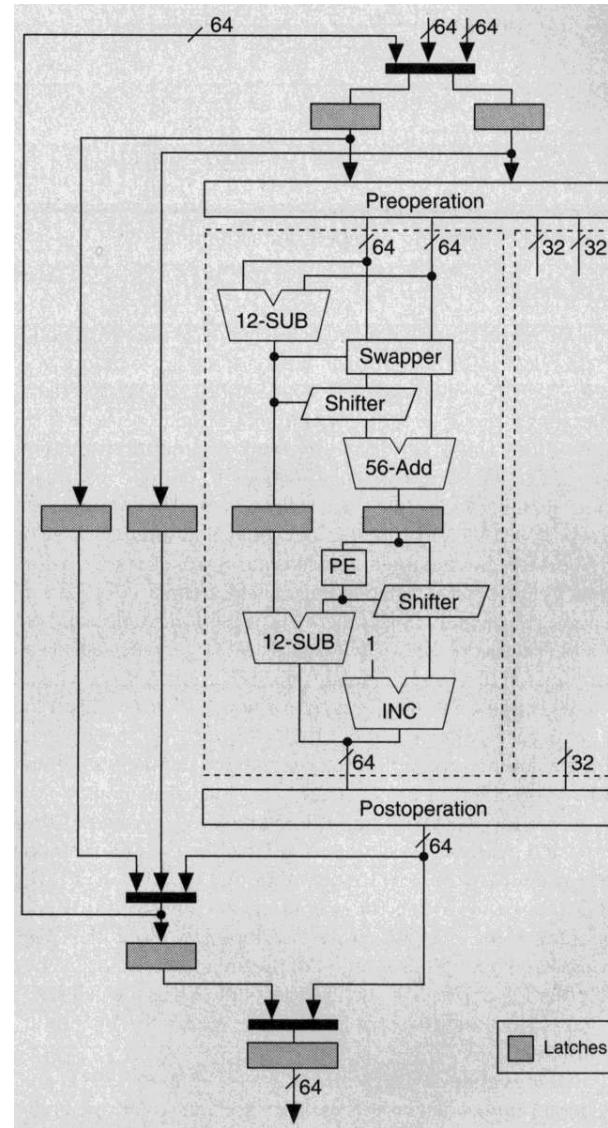
- Coprocesseur vectoriel 64 bits
- 50 MHz
- 128 Ko de registres vectoriels
- 64 octets de masque
- superscalaire : INT+INT+FP ou INT+FP+FP
- 206 MFLOPS SP, 106 DP

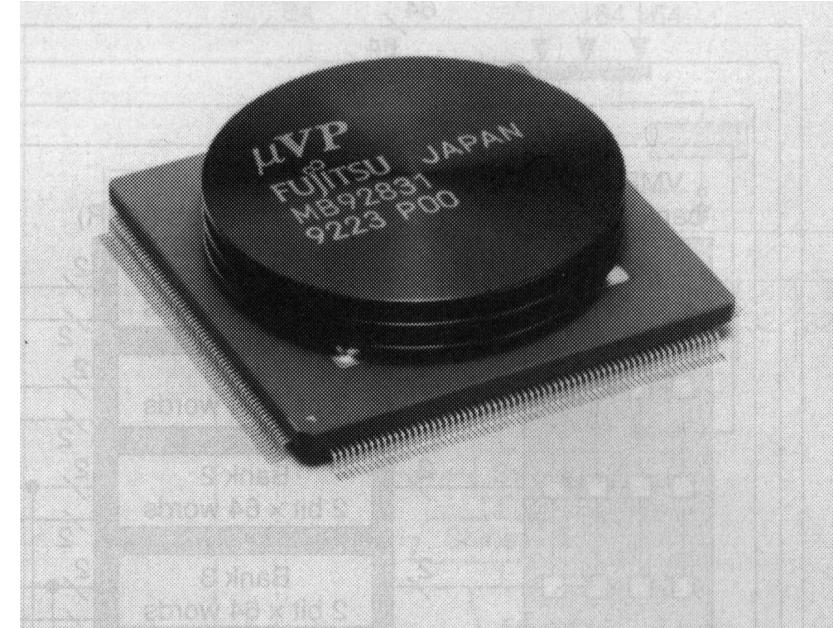
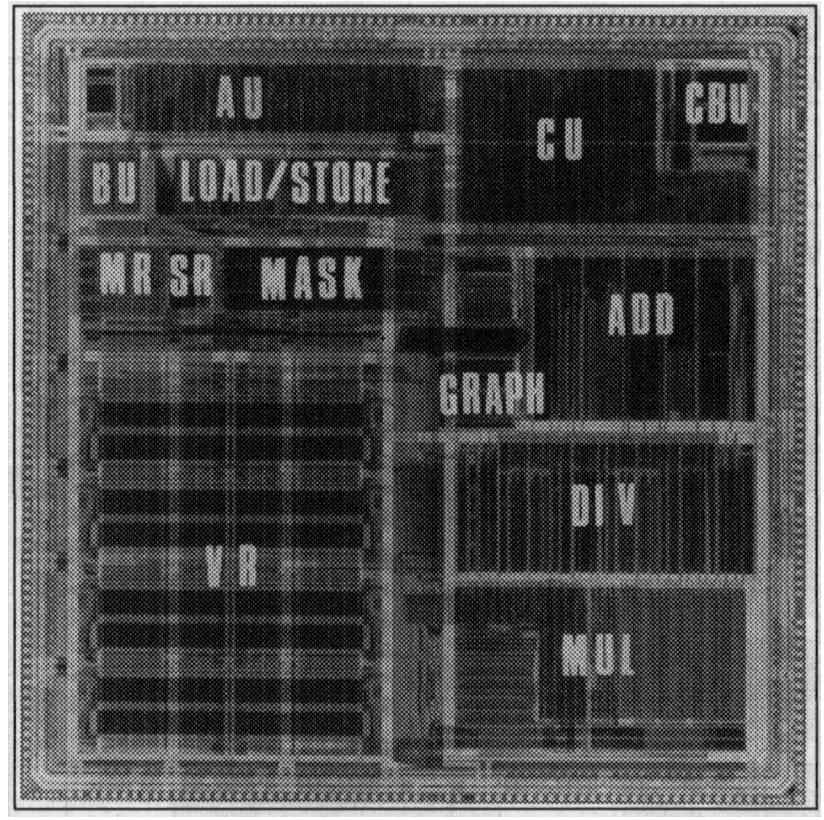












- *feed-forwarding* : matériel en  $\mathcal{O}(p)$
- gel du pipeline : comparer la destination d'un chargement avec les sources des instructions suivantes.  $\mathcal{O}(p)$  comparateurs + *feed-forwarding* du résultat.



Le pipeline est visible au programmeur  $\approx$  micromachine.  
Attention à ne pas utiliser ce qui n'est pas encore calculé !

Exemple du i860 : avance du pipeline *explicite*

```
pfadd f1,f2,f3
```

- met dans f3 ce qui sort à l'instant  $t$  de l'additionneur flottant
- met à l'instant  $t$  f1 et f2 à l'entrée de l'additionneur.

Approche « RISC » de tout gérer à la compilation mais :

- compilateurs encore mauvais
- pour le (micro)programmeur :  ...



1. Détection tardive d'un branchement
2. Détection encore plus tardive du résultat d'un test.

Si 30 % de branchements faisant perdre 3 cycles :

$$\frac{a}{a_{\text{idéal}}} = \frac{1}{1 + 0.3 \times 3} \approx 0,53$$

Syllogisme informatique :

Machine rapide =: Pipeline =: branchements lents =: machine lente...

Solutions :

1. rajouter du matériel pour accélérer le branchement
2. parier sur le résultat d'un test.



En même temps que le décodage, rajouter dans ID :

- additionneur pour le calcul de l'adresse au lieu de l'additionneur du processeur
- logique testant un registre et/ou une condition.



En moyenne :

- branchements conditionnels plus nombreux que les branchements non conditionnels
- VAX :
  - ▶ 50 % des branchements sont pris
  - ▶ tests de bit : plutôt non pris
  - ▶ dans les boucles : pris à 90 %
- en général les branchements en arrière sont pris (boucles).



Matériel en sorte qu'on :

- suppose que le branchement est toujours pris : commencer IF le plus tôt possible
- suppose que le branchement n'est jamais pris : IF standard PC++.



Incapable d'exécuter un branchement rapidement ?  
Changer la sémantique !

```
branchement
 suivante_1
 suivante_2
 ...
 suivante_n
 cible_si_branchement_pris
```

Les  $n$  instructions suivant un branchement sont *systématiquement* exécutées.

=: compilateur et utilisateur doivent en tenir compte !

Attention si branchement dans une instruction suivante...



Compilation :

1. mettre une instruction précédente si possible. Si oui : meilleure solution
2. une instruction de la cible si l'instruction peut être « défaite »  
   $\Leftarrow$ : prédire que le branchement sera pris
3. une instruction de la suite si l'instruction peut être « défaite »  
   $\Leftarrow$ : prédire que le branchement ne sera pas pris.

Dépend beaucoup du compilateur !



Se rappeler le passé :

- Quel était le dernier branchement ?
- Quel était le dernier branchement à cette adresse ?
- Quels étaient les derniers branchements à cette adresse ?
- Adresse cible du dernier branchement à cette adresse ?
- Quelle était la cible du dernier branchement à cette adresse (29000) ?

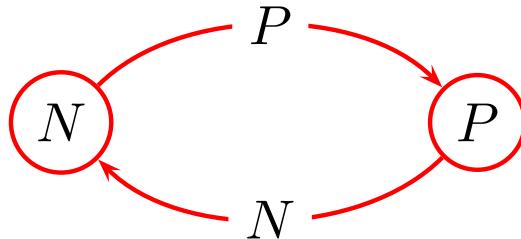
Exemple : ajouter des bits de prédiction au cache d'instructions (ALPHA 21064).



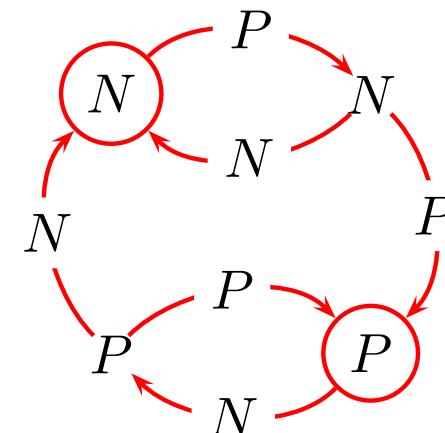
```

for ( i = 0; i < N; i++) {
    if (pas souvent) {
        /* Rarement... */
        ...
    }
    ...
}

```



Prédiction à 2 bits :  
branchements à l'*intérieur* de  
boucle ↵ se tromper qu'1 fois.



Problèmes liés au pipeline :

- quelle instruction a tiré la sonnette ?
- plusieurs exceptions simultanées ou presque
- nécessité de relancer la machine dans l'état au retour d'exception
- sauvegarde de *plusieurs* PC
- l'arrêt n'est pas toujours assez rapide

Interruption précise si aucune instruction suivant l'instruction fautive n'est exécutée. Plus simple pour le débogage.



- propager dans le pipeline avec chaque instruction son status d'exception et traiter dans l'ordre de sortie
- traiter les exceptions dès qu'elles surviennent :
  - ▶ arrêter toute écriture
  - ▶ regarder le code pour savoir quelle instruction a pu déclencher telle exception à l'instant  $t$ .



Jusqu'à présent :

- instructions exécutées dans l'ordre
- si une est bloquée elle bloque les suivantes.

Planification dynamique :

- dépendances non connues à la compilation (branchements conditionnels et cibles)
- exécution dans le désordre
- ↵ fin d'exécution dans le désordre !
- portabilité de code entre différents pipelines



Exécution dans l'ordre mais opérateur de longueur quelconque (flottant).

Attendre dans EX : trop mauvais ! ↵ Besoin :

- tester conflit structurel : bits d'utilisation unités fonctionnelles (*scoreboard*)
- tester RAW/WAW : bits d'utilisation registres (*scoreboard*)
- gérer le *feed-forwarding*

Exceptions précises : difficile !

- fichier d'historique des registres
- stocker de l'information sur les instructions arrêtées.

Si limite matérielle, diminuer le nombre d'instructions //,



Supercalculateur CDC 6600 [Thornton, 1964] :

- Instructions RISC
- Essaye de lancer 1 instruction par cycle
- Profite de plusieurs unités fonctionnelles
- Recouvrement des exécutions
- Gestion des dépendances de données : « tableau noir ».  
Attention à la compilation...
- Fin d'exécution dans le désordre
- Alimentation : pile d'instruction
- 5 accès à la mémoire simultanés
- Mémoire de 32 bancs de 4kmots de 60 bits.



Supercalculateur IBM 360/91 [Tomasulo, 1967] :

- accélérer du code IBM 360
- code registre-mémoire
- seulement 4 registres flottant
- mémoire à forte latence
- opérateurs à forte latence
- opérateurs pipelinés (vus comme plusieurs opérateurs par l'algorithme)
- ↵ déroulage matériel des boucles.



Différence par rapport au CDC 6600 :

- algorithme distribué
  - feedforwarding sur 1 bus
  - registre contient une valeur ou le numéro de l'unité devant lui donner sa valeur
1. instruction → réservation ou tampon libre (accès mémoire) sinon contention structurelle. Si opérande dans registre, lire l'opérande sinon envoyer le nom de l'unité qui le fournit
  2. si opérande non disponible, on attend que le résultat passe sur le CDB
  3. écrire le résultat sur le CDB et dans le registre.



- Déroulage des boucles même avec peu de registres
- Load/Store = opérateurs.

Inconvénient :

- CDB = goulet d'étranglement
- complexité en  $\mathcal{O}(\#CDB)$
- dépendances au niveau de la mémoire : nécessite de comparer les adresses de Load/Store.



Si dépendance lecture-écriture

```
a = b;  
b = c + d;  
e = f(b);
```

Exécuté en interne (table de renommage)

```
a = b; b' = c + d;  
e = f(b');
```

Mécanisme actuel plus actuel (IBM Power : 6 registres en plus des 32 visibles)



Exploitation à la compilation du parallélisme des boucles :

- économise du contrôle de flot
- suppression des WaR en renommant les registres
- supprimer des RaW en réarrangeant le code pour écarter les instructions dépendantes
- Nécessite beaucoup de registre et/ou du renommage



Ressemble à un pipeline matériel :

Boucle initiale  $\rightsquigarrow$  boucle contenant des instructions appartenant à différentes itérations.

1 itération finale  $\approx$  1 cycle de pipeline.

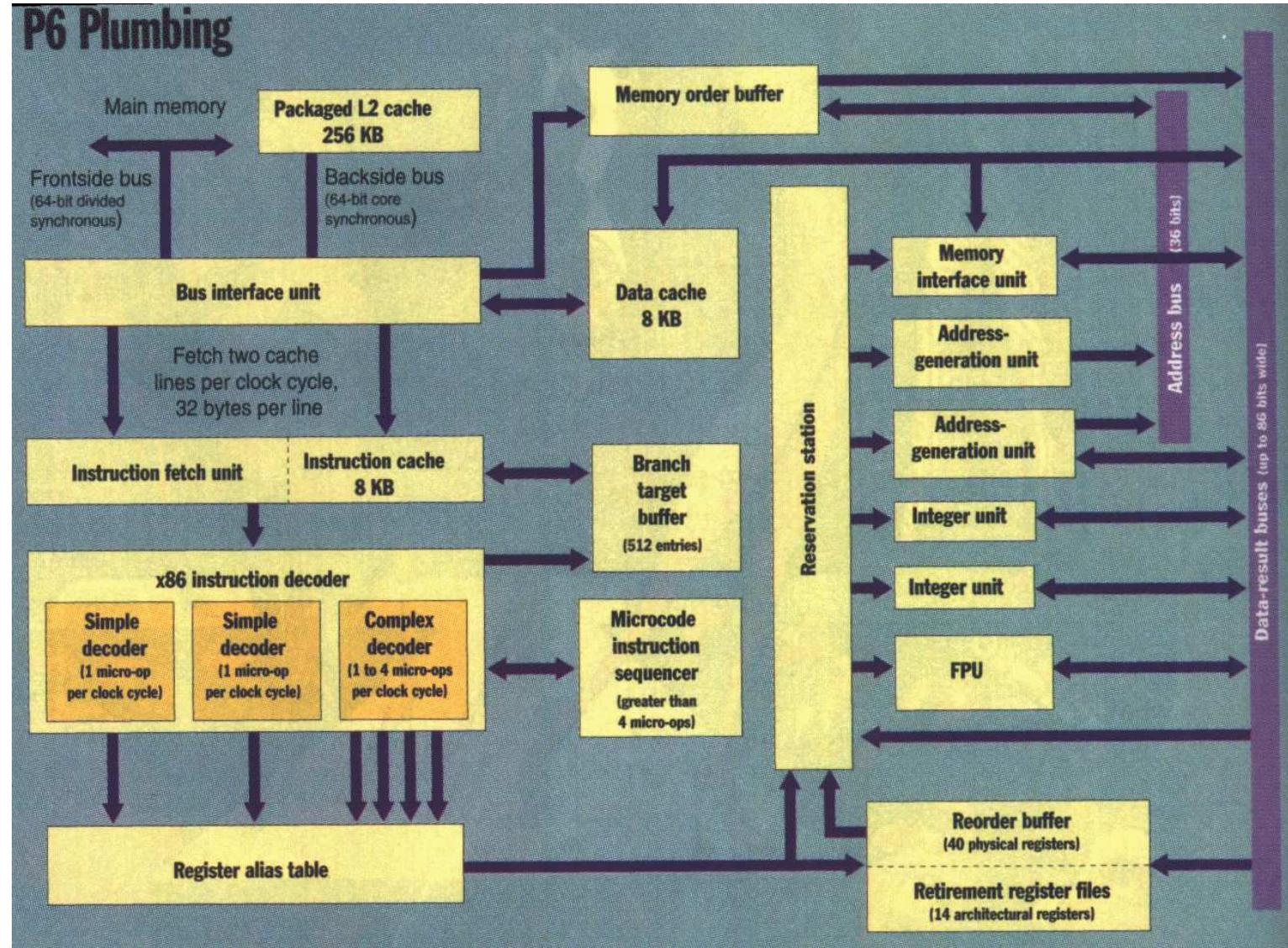
Exemple : pipeline flottant du i860.

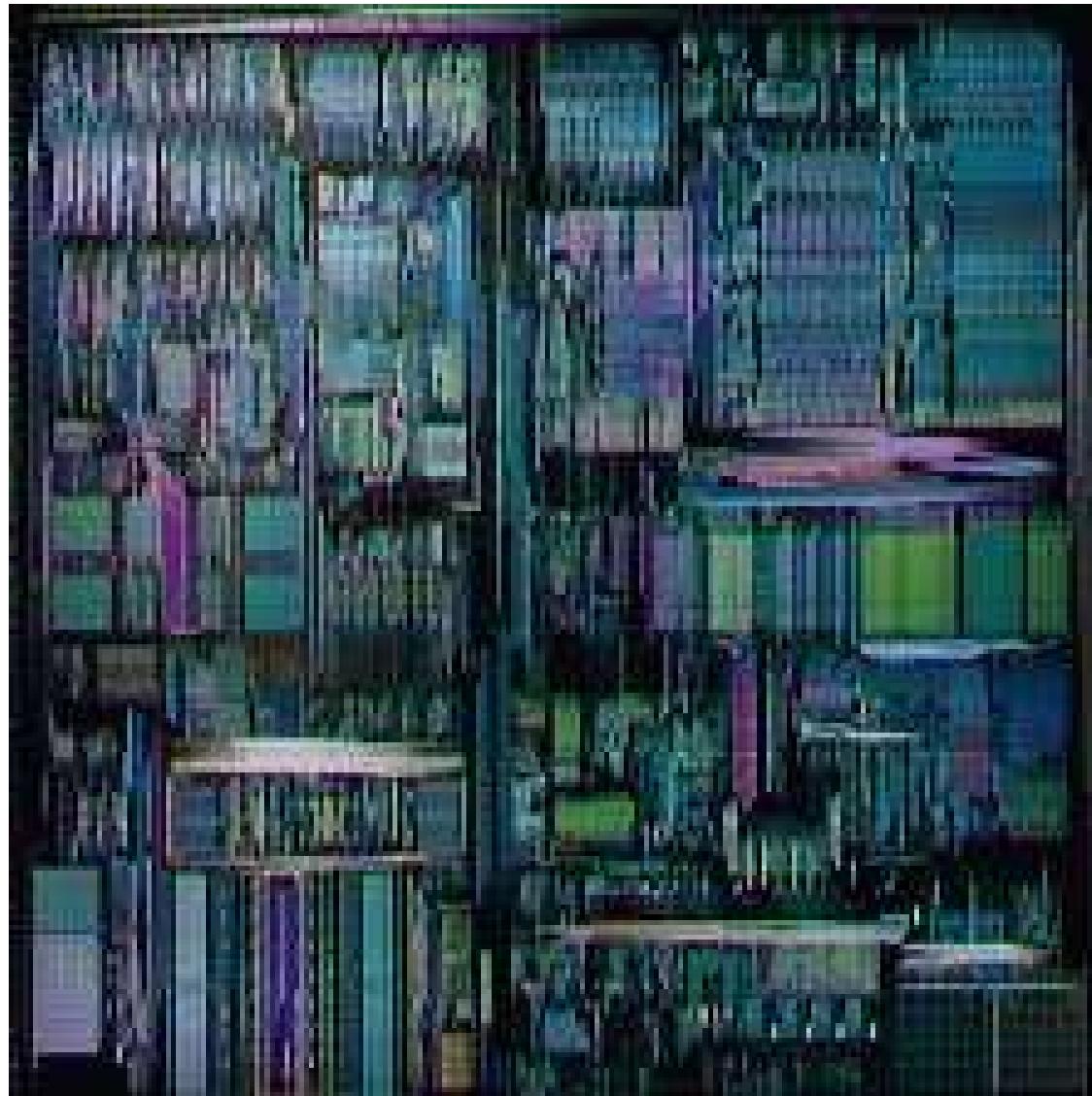


Dépasser une instruction par cycle en lançant plusieurs instructions par cycle [Tjaden and Flynn, 1970] [Riseman and Foster, 1972] :

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







Extension :

- suit les 2 chemins d'un branchement et défait le travail inutile  
(IBM 360/91, MC88110)

```
if (a > 0) then
```

```
    b = a
```

```
else
```

```
    b = -a
```

- difficile de défaire la mémoire ↪ souvent bloquant sur les écritures en mémoire
- éliminer les exceptions des chemins non réellement exécutés (division par 0,...)
- suit beaucoup de chemins [Riseman and Foster, 1972]

$N$  branchements =:  $2^N$  « machines ».

► Problème intrinsèque au superscalaire : matériel exponentiel pour lancer les instructions [Tjaden and Flynn, 1970] ☺

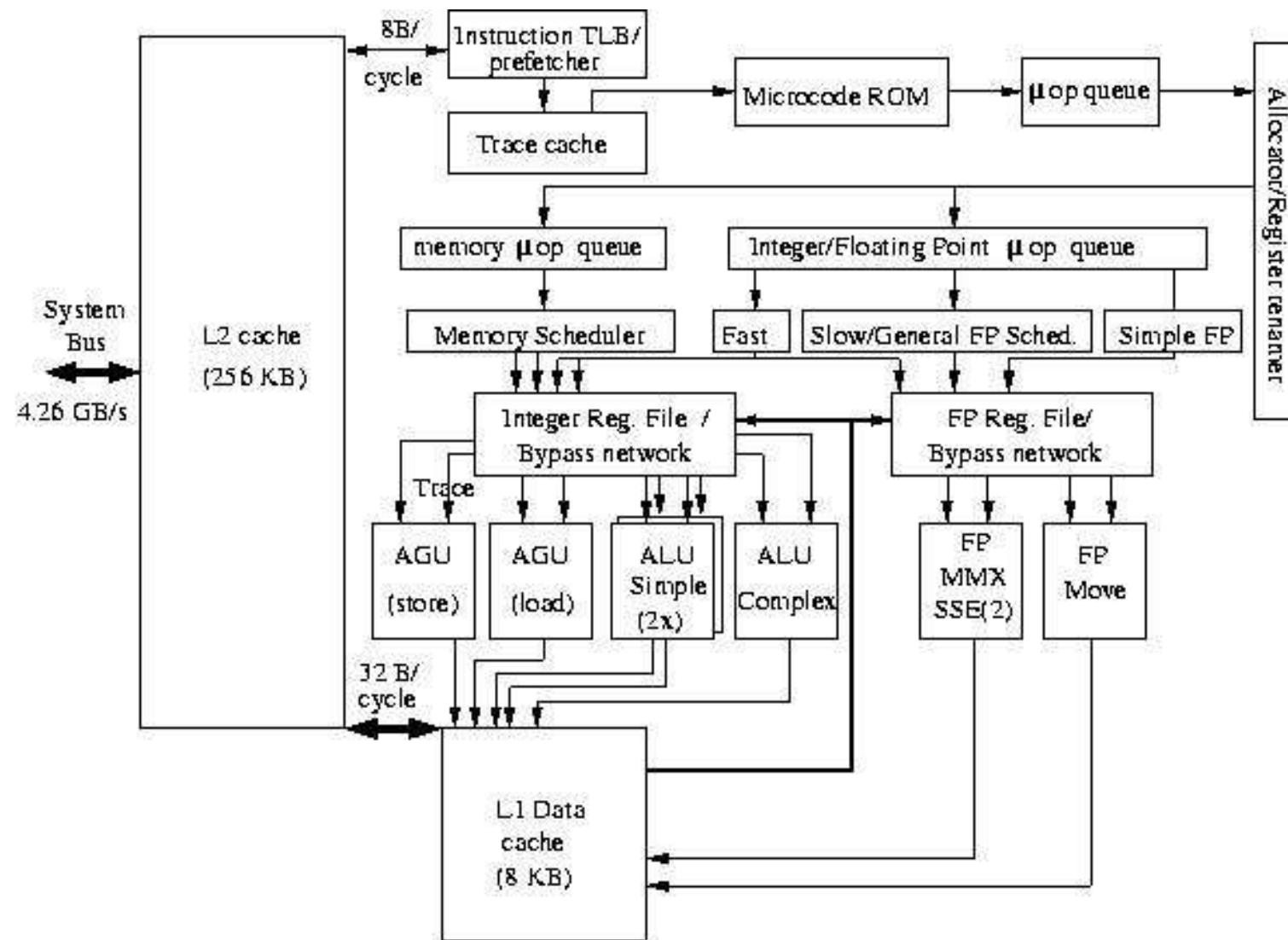


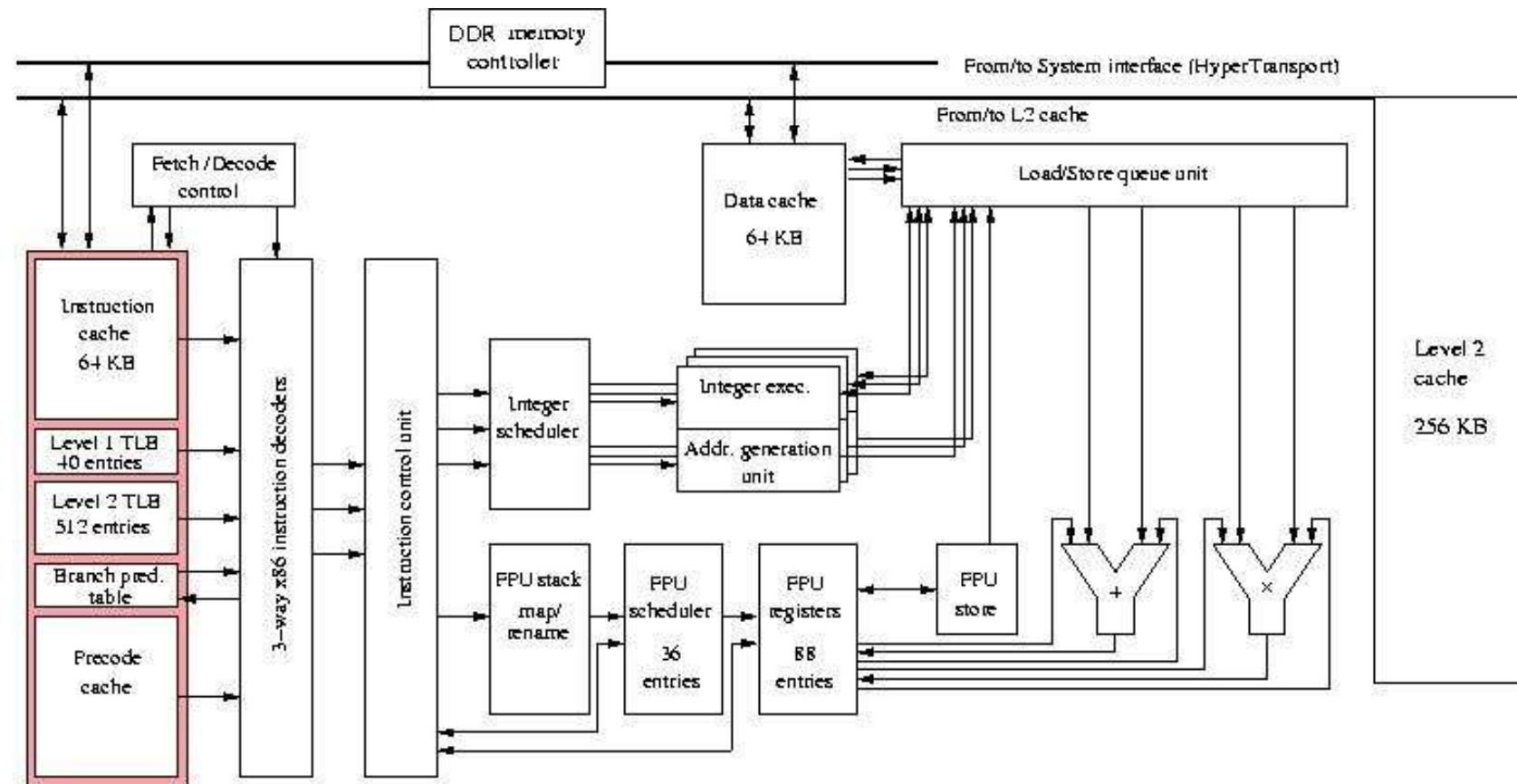
- ▶ Mais loi de Moore aussi ☺



- Pentium et Athlon récents : traduit chaque opération CISC en des opérations RISC exécutées par un superscalaire RISC
- Pentium 4 : cache de trace  
Cache les instructions RISC traduites pour éviter traduction et planification



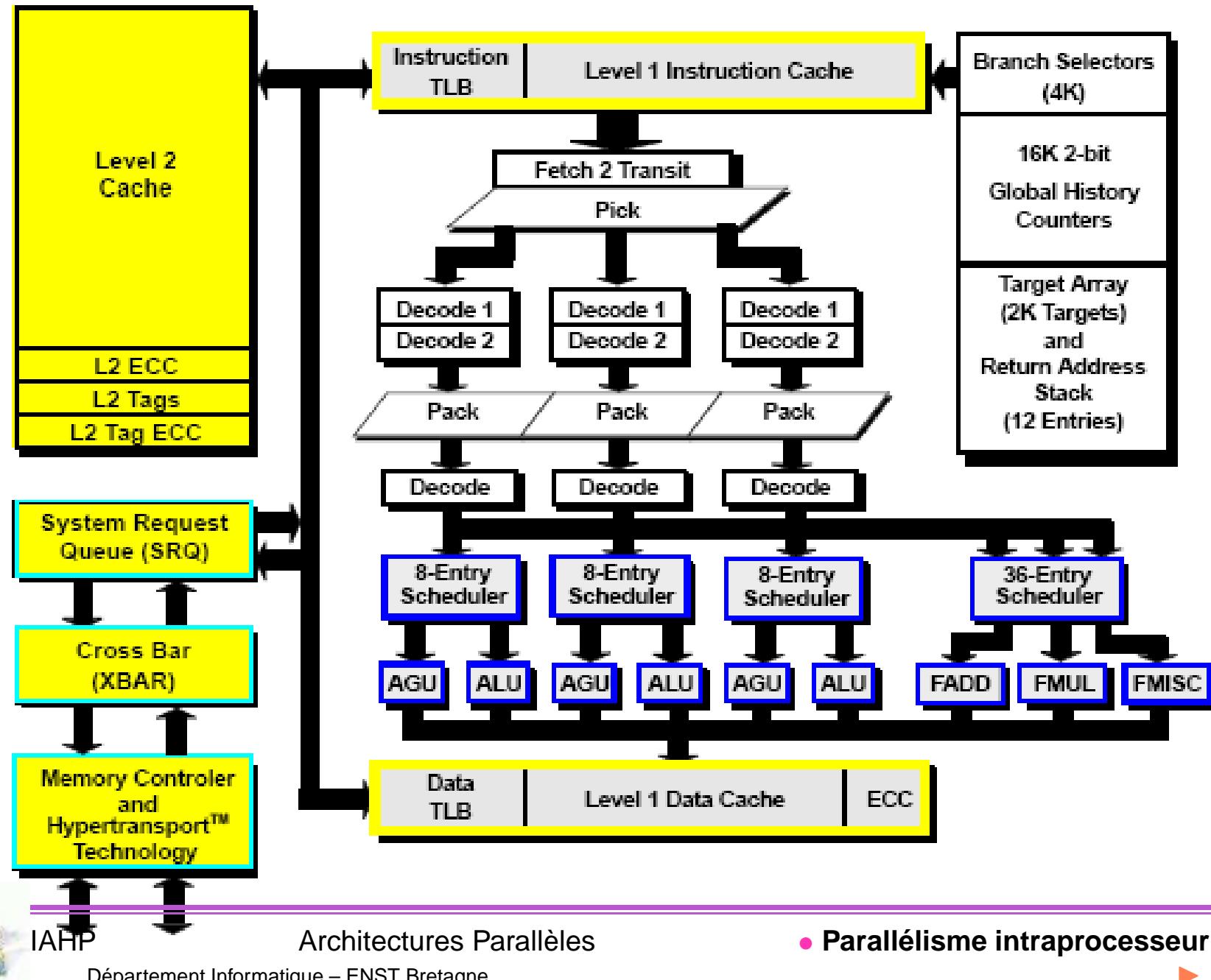




- Jeu d'instruction à la *x86*
- Mode 64 bits qui double aussi nombre de registres
- Superscalaire à exécution dans le désordre de 9 instructions/cycle
  - ▶ 3 instructions entières
  - ▶ 3 générations d'adresses
  - ▶ 3 calculs flottants (add, mul, mémoire)
- Interface par 3 canaux HyperTransport de 8 Go/s au monde extérieur (mais  $0,1 \times$ SX-8)

[http://www.amd.com/us-en/assets/content\\_type/white\\_papers\\_and\\_tech\\_docs/251](http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/251)





## Extension zen du RISC

~~~ Very Long Instruction Word

- Plusieurs unités fonctionnelles programmées explicitement
(programme = microprogramme !)
- Mémoire de code pas chère est rapide
- Progrès des compilateurs à partir des années 70
[Foster and Riseman, 1972, Fisher, 1981]
- Planification statique du code pour remplir les unités

| | | |
|-------------|-------------|-----------|
| $a = b + c$ | | |
| $d = e + f$ | $a = b + c$ | $g = h*i$ |
| $g = h*i$ | $d = e + f$ | $j = k*l$ |
| $j = k*l$ | | |



- Simple à réaliser
- Donc rapide
- Compliqué à programmer efficacement (bon compilateur)
- Les applications ressemblent assez au vectoriel
- Gel de pipeline = : gel de tout !

i860 (1987) : 2 instructions par cycle (entier + flottant)

Peu de succès et performances mitigées... Vers un grand retour
(Merced/Itanium IA-64 HP+Intel) ?



- Bi cœur : 10^9 transistors...
- Pipeline et exécution *dans l'ordre* de 6 instructions/cycle
- 128 registres entiers, 128 registres flottants, 128 registres prédictifs/conditions visibles
- Contrôle très fin de la micromachine avec VLIW EPIC
- Encore plus de stress sur le compilateur (et programmeurs ☺)
- Mais permet d'avoir de bonnes performances



- Suppression des branchements (technique introduite dans les machines SIMD [Slotnick et al., 1962]) :

if (A>0)

B = A; cond1 = (A>0)

else if (cond1) B = A

B = -A; ↵ if (!cond1) B = -A

Méthode d'*if-conversion* introduite dans la vectorisation [Allen et al., 1983] [Kennedy and McKinley, 1990]. L'imbrication est gérée par des gardes plus complexes[Keryell and Paris, 1993]

Nécessite un contrôle d'exécution de chaque opérateur

Introduction du MV reg1,reg2 conditionné dans les RISC (ARM, ALPHA, UltraSPARC) et autres (PentiumPro,...)

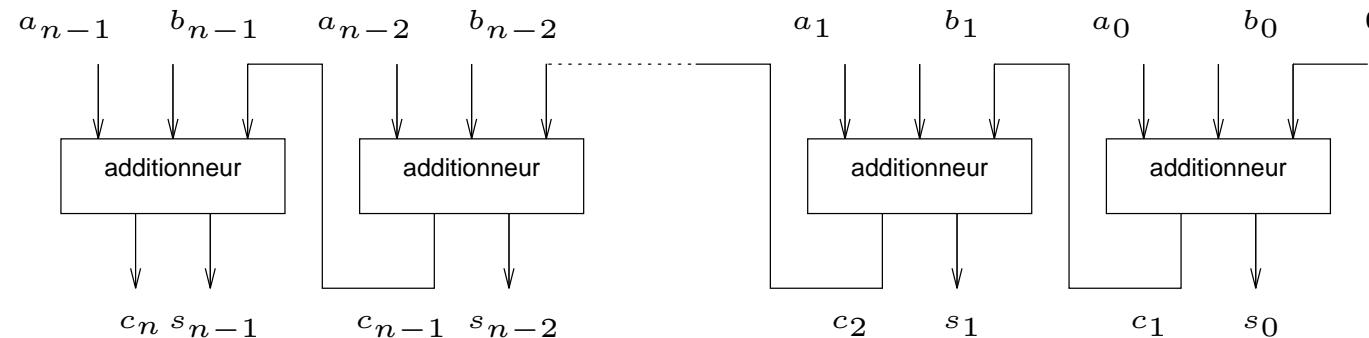
- Remplacer les branchements par autre chose... Instructions de boucles (DSP, Itanium 2,...)



Additionneur 1 bit : $(c,s) = \text{add}_1(a,b) = (a \wedge b, a \oplus b)$ (4 transistors)

Additionneur complet (3 entrées) :

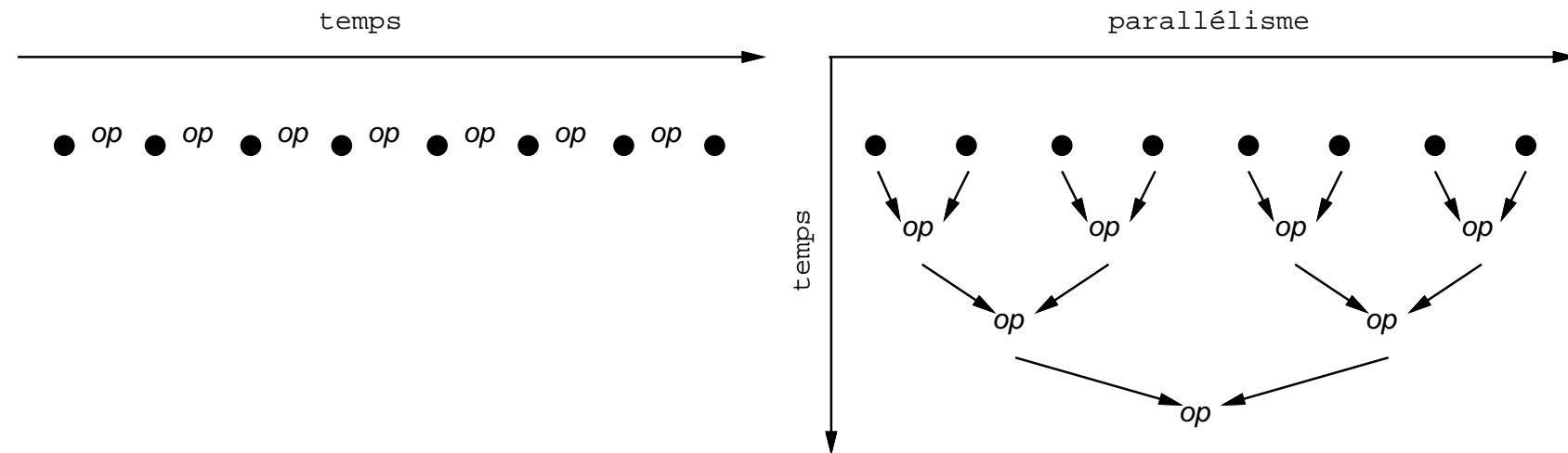
$$(c_o, s) = \text{add}(a, b, c_i) = ((a \wedge b) \vee (a \wedge c_i) \vee (b \wedge c_i), a \oplus b \oplus c_i)$$



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



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



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

gâcher utile !

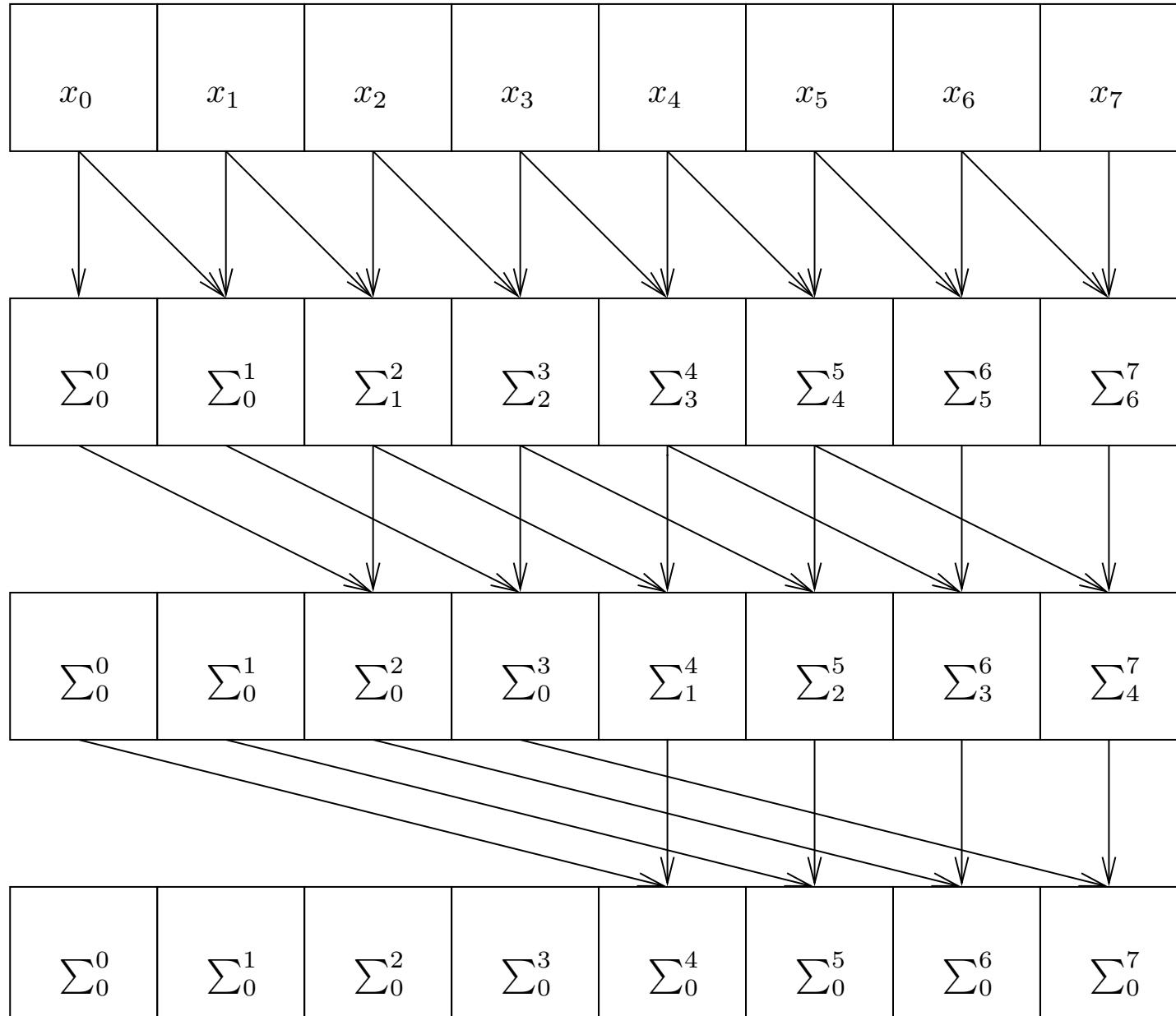
Parallélisme :



Autre algorithmique parallèle classique [Barnes et al., 1968] :

$$\forall i \in [0, n - 1], \quad S_i = \bigoplus_{j=0}^i x_j$$





Si réutilisation des opérateurs :

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



non associativité des opérations flottantes... Changement de l'ordre d'évaluation

~~> changement du résultat



Retenue = facteur limitant \rightsquigarrow changements de variable :

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

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

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

Appliquer une opération parallèle préfixe : temps en $\mathcal{O}(\log n)$ et espace en $\mathcal{O}(n \log n)$.

Autres méthodes : *carry skip*, *carry select*...

Soustraction : inverser une des entrée (\bar{a} ou \bar{b}) et $c_0 = 1$



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

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

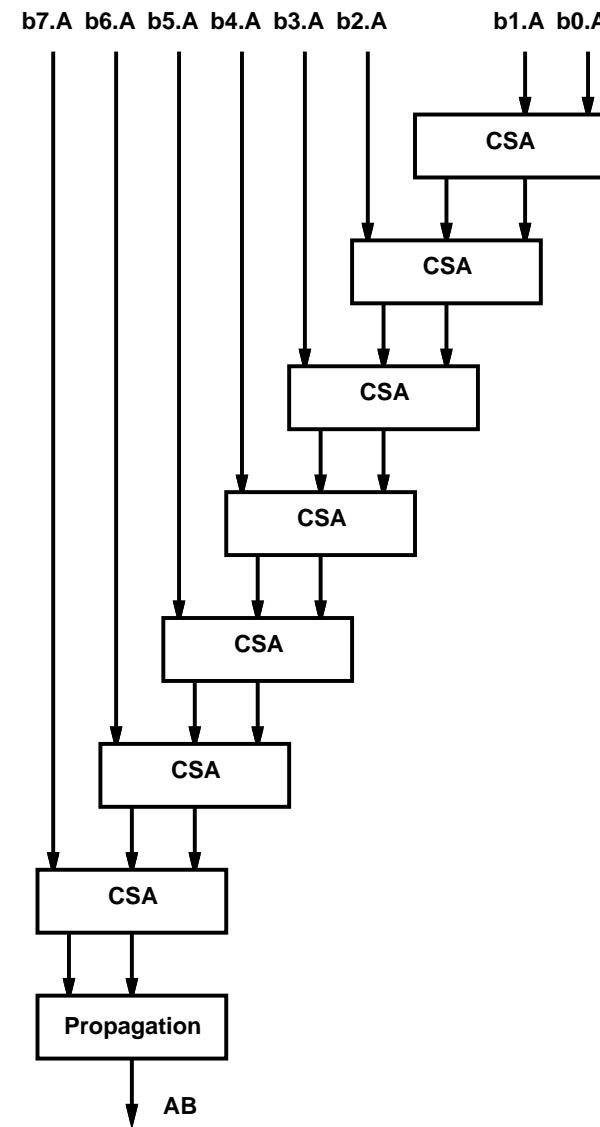
Bonne vieille méthode manuelle : n additions :

- Sauter les bits à 0 (temps de multiplication non constant)
- Propagation de la retenue lente ↗ ne pas propager et garder toutes les retenues (*Carry Save Adder*) et propager lors de la dernière addition

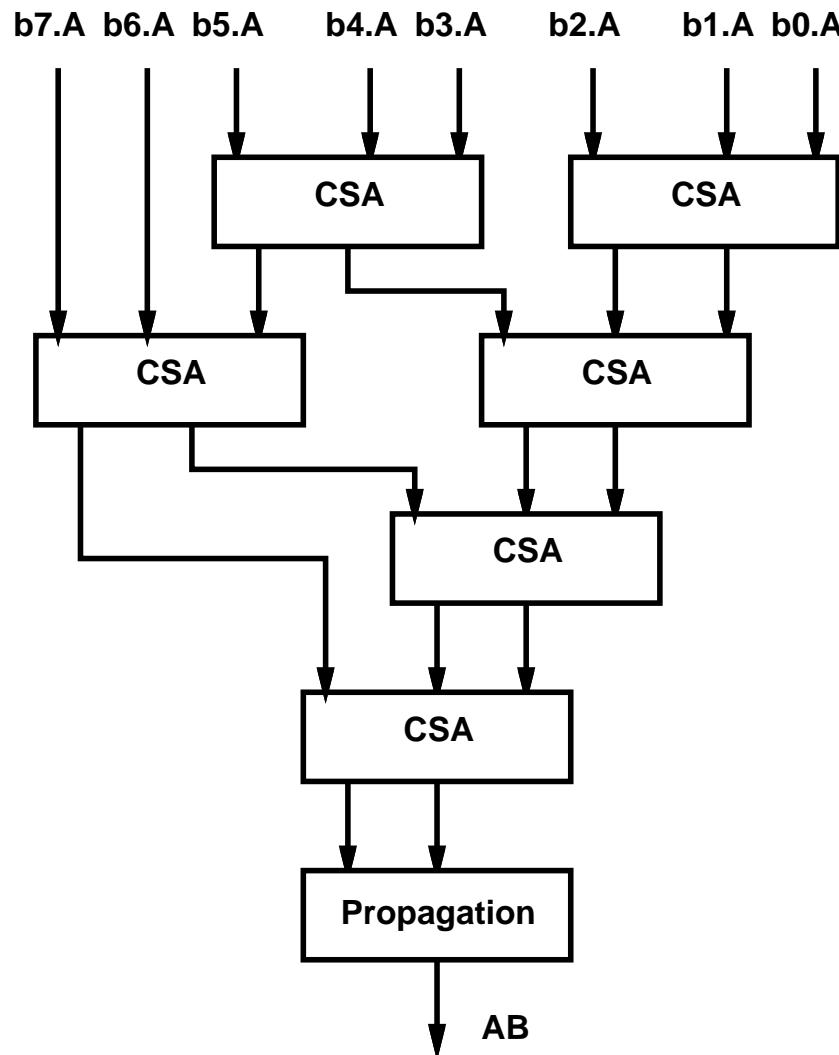
Utilisation d'un codage redondant ($2 \times n$ bits, chiffres dans 0–3) sauf pour le dernier résultat :

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





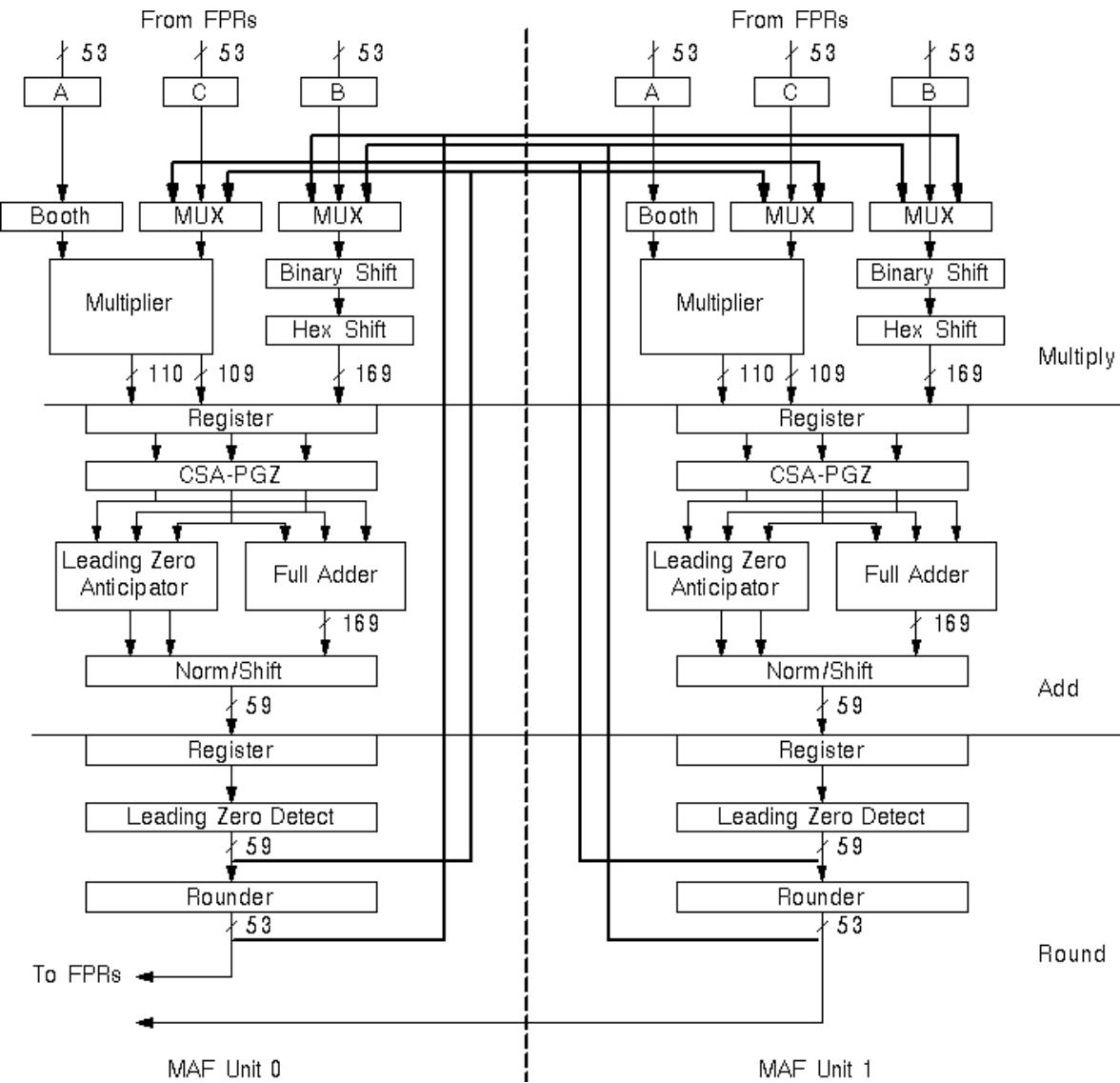
Réduction ↘ arbre :



Calcul scientifique : produit scalaire ↗ 1 + & \times par cycle

Power 2 (SP2) (« + » & « \times ») par cycle ↗ Bon compilateur et programme adapté...





| Machine/Processeur | SPECint95 |
|-----------------------------|-----------|
| DEC Alpha 21164 à 600 MHz | 18,8 |
| HP-PA 8200 à 236 MHz | 17,3 |
| SGI R10000 à 250 MHz | 14,7 |
| IBM PowerPC604e à 360 MHz | 14,2 |
| Sun UltraSPARC II à 300 MHz | 12,3 |
| Intel Pentium II à 300 MHz | 11,7 |

Quelques résultats récents de SPECint95 (Source
<http://www.specbench.org/osg/cpu95/results/cint95.html>)

Mars 2003 :



| Machine/Processeur | SPECCINT2000 | SPECCFP2000 |
|----------------------------------|--------------|-------------|
| IBM Power4 à 1,3 GHz | 814 | 1169 |
| Athlon XP 1900+ | 711 | 634 |
| Intel Pentium 4 Xeon 2 GHz | 663 | 734 |
| DEC Alpha 21264C EV68 1 GHz | 679 | 960 |
| Sun UltraSPARC III Cu à 1050 MHz | 610 | 827 |
| HP PA-8700 à 750 MHz | 604 | 576 |
| SGI R14000 à 500 MHz | 427 | 463 |

Mars 2004 :



| Machine/Processeur | SPECCINT2000 | SPECCFP2000 |
|---------------------------------|--------------|-------------|
| IBM Power4 à 1,7 GHz | 1158 | 1776 |
| AMD Athlon 64 3200+ | 1335 | 1250 |
| AMD Opteron 148 | 1477 | 1514 |
| Intel Pentium 4 EE 3 GHz | 1705 | 1556 |
| Intel Pentium 4 Xeon 3.2 GHz | 1563 | 1347 |
| Intel Itanium2 1,5 GHz | 1404 | 2119 |
| DEC Alpha 21264C 1,25 GHz | 845 | 1365 |
| DEC Alpha 21364 1,15 GHz | 795 | 1482 |
| Sun UltraSPARC III Cu à 1,2 GHz | 722 | 1344 |
| HP PA-8700+ 875 MHz | 678 | 674 |
| SGI R14000 600 MHz | 500 | 529 |



<http://www.spec.org>



Parallélisme intraprocesseur :

- Difficile à faire sans registres (pile \equiv goulet, mémoire \equiv goulet)
Registres = parallélisation de l'accumulateur et optimisation logicielle !
- Registre de codes de condition à partager
- Taille des circuits intégrés finie...
- Tant que la mémoire est à l'extérieur : goulet
- De plus en plus compliqué de concevoir un processeur...
- Penser parallélisme intraprocesseur avant interprocesseur
- Maintenant plusieurs processeurs/circuit

Le RISC simplifie bien les choses...



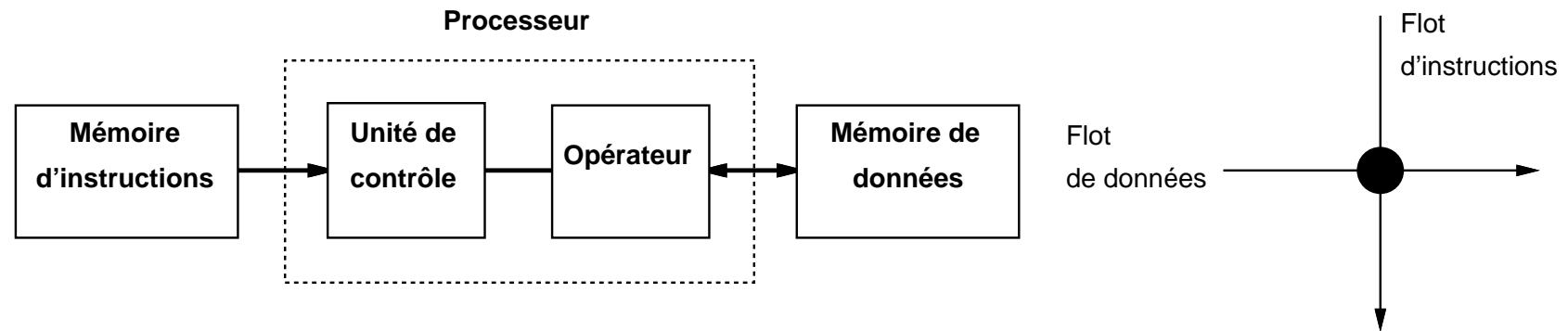
- quand on a exploité les limites du parallélisme intraprocesseur...
- 1975 : introduction du microprocesseur, « parpaing » de l'informatique & l'électronique
- loi de GROSCH modifiée : mieux vaut plusieurs ordinateurs plus petits de technologie moins chère



Taxinomie assez grossière (1966) mais représentative [Flynn, 1966] :

$$\{S,M\} \times \{I\} \times \{S,M\} \times \{D\}$$

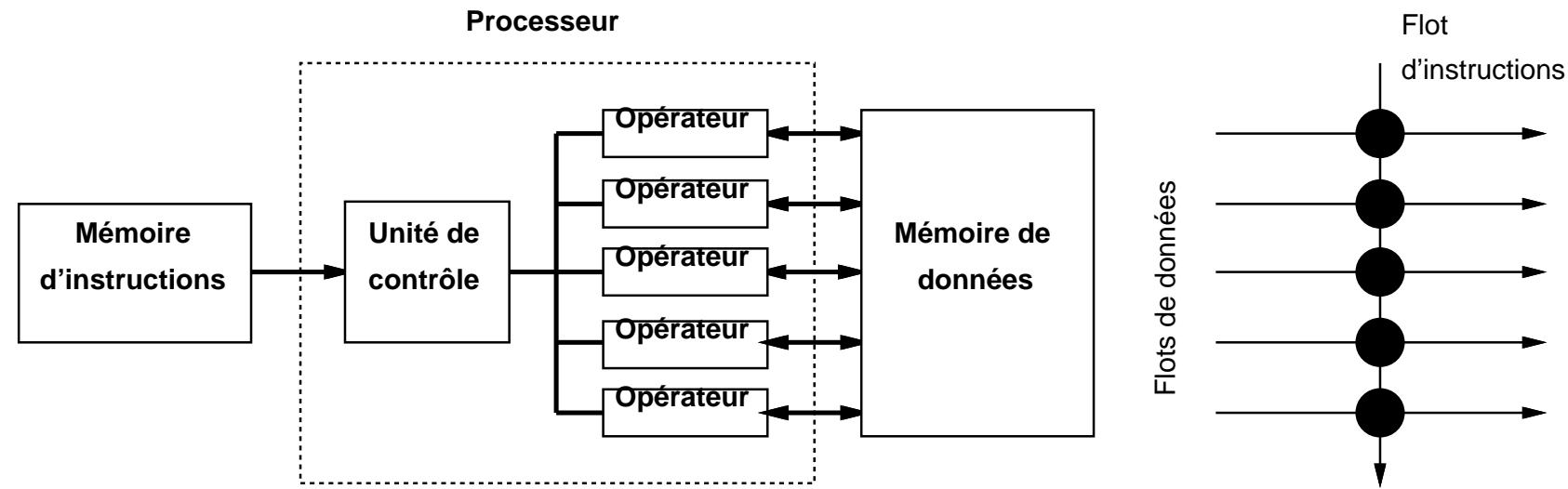
SISD : *Single Instruction stream Single Data stream*



séquentiel classique

SIMD : *Single Instruction stream Multiple Data stream*

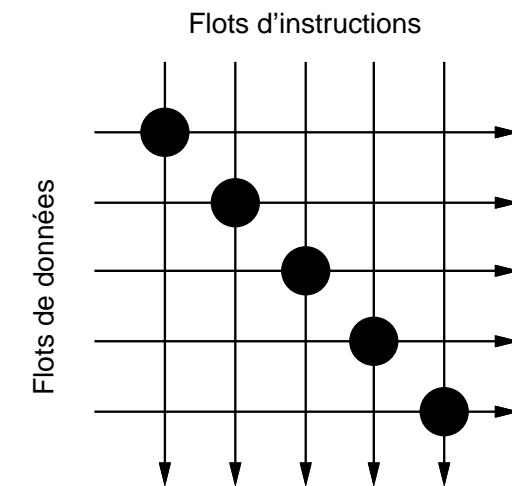
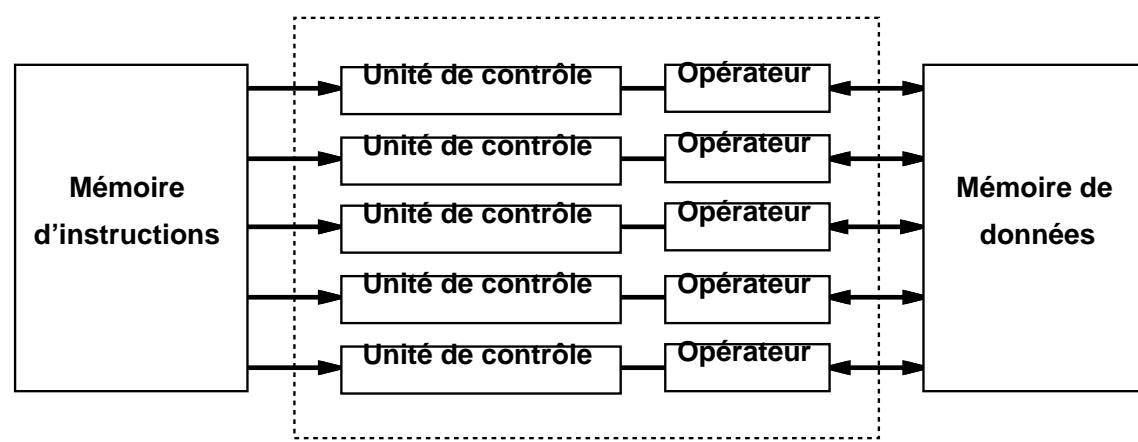




extension logique : travail sur des vecteurs ou matrices. VLIW aussi...



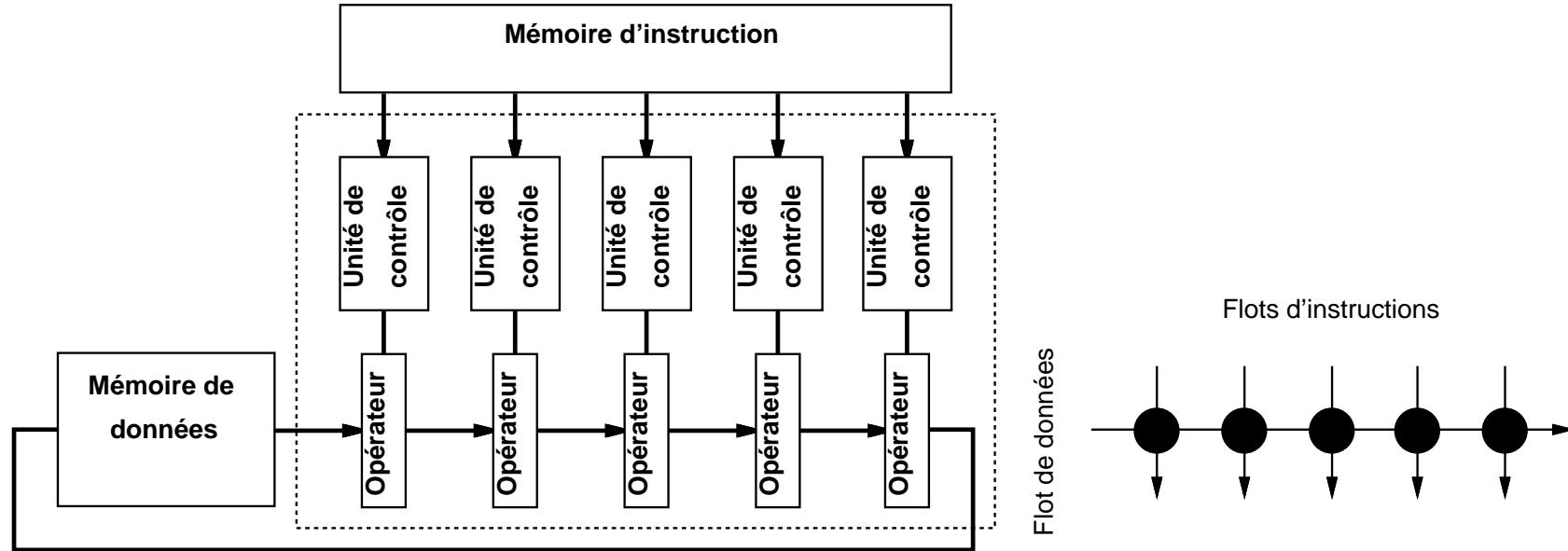
MIMD : *Multiple Instruction stream Multiple Data stream*



toutes les libertés

MISD : *Multiple Instruction stream Single Data stream*





flot de données, traitement du signal, macropipeline ?



- Extension simple au modèle SISD : plusieurs flots de données
- Opérations vectorielles élément par élément rapides
- Modèle à parallélisme de données très simple

$$\begin{matrix} C \\ = \\ \boxed{\begin{array}{|c|c|c|c|}\hline & & & \\ \hline \end{array}} \end{matrix} + \begin{matrix} A \\ \boxed{\begin{array}{|c|c|c|c|}\hline & & & \\ \hline \end{array}} \end{matrix}$$

- Contrôle et code factorisés : logique pour applications numériques
- Processeurs + simples
- 1 contrôleur d'instructions ↗ rigidité du comportement

```
if (a > 0) then  
    b = a  
else  
    b = -a
```



- Synchronisation globale ↗ pas de temps de synchronisation
- Synchrone ↗ impossible d'utiliser des caches
- *synchronisme pessimiste*
- SIMD avec recouvrement du séquencement : plus simple que du vectoriel
- *Processeurs spécialisés*



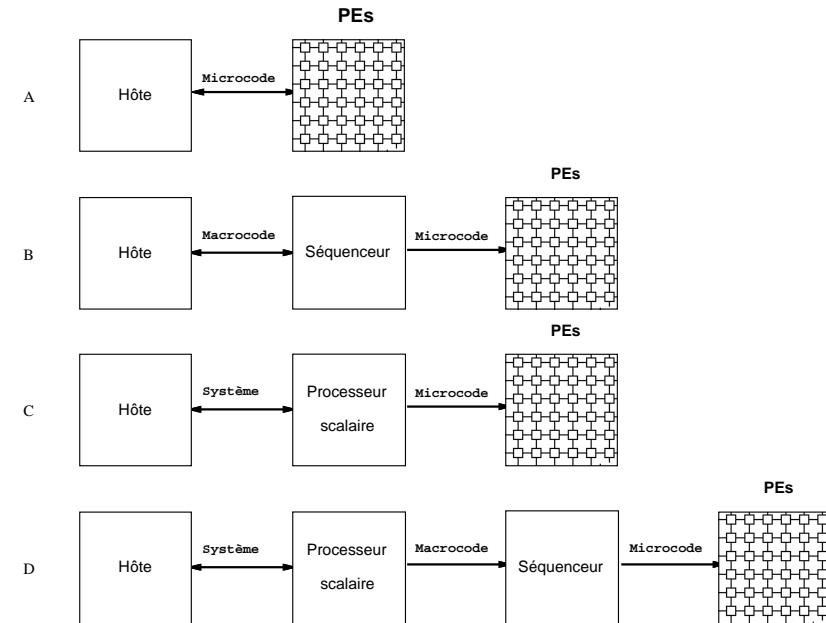
Hôte : exécute l'environnement système

Processeur scalaire : exécute la partie scalaire du programme

Séquenceur : transforme le code parallèle en microcode parallèle (virtualisation,...)

Pour des raisons de coût :

projections possibles sur moins de matériel.

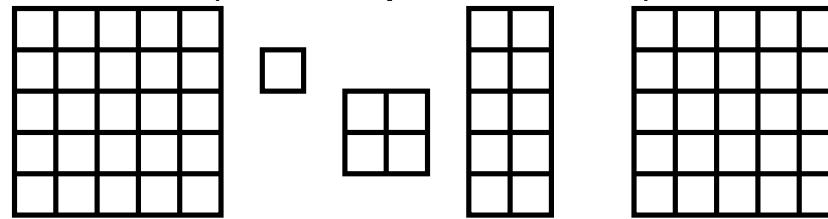


- Système permettant de contrôler l'activité pour l'*if-conversion*
- Adressage local indexé difficile si PE à grain fin : manipulation de grosses adresses !
- Système pour diffuser des valeurs scalaires à tous les PEs
- possibilité de récupérer une valeur globale : « *ou global* »
- Souvent PEs à grain fin, mais efficacité ↴...



Pallie le manque d'individualité du SIMD au coût de certaines synchronisations (rendez-vous)

- Programmation libre mais plus compliquée
- Plusieurs problèmes indépendants possibles (UNIX)
- Parallélisme de tâches (\pm indépendantes)



- Interbloquages possibles
- Alimentation en code plus coûteuse mais **caches** (désynchronisation)
- Synchronisation pouvant être coûteuse ↗ matériel pour accélérer les synchronisations :
 - ▶ iPSC/860 : $\approx 1,5$ ms



- ▶ CM-5 : $\approx 1 \mu\text{s}$ (réseau de synchronisation)
- *Processeurs standard*
- Nombreux modèles de programmations : tous ! Processus communiquants, CSP, fonctionnel,... mais avenir là aussi du parallélisme de données
- Collection d'ordinateurs « standard » ↗ solution d'avenir probable



Récupération des résultats de l'algorithmique répartie.

Barrière de synchronisation : bloquer les processeurs tant que tous ne l'ont pas atteinte.

Réalisation : condition globale calculée par un arbre de réduction ou « collecteur ouvert ».

Optimisation : barrière floue.

Séparation de la barrière en entrée barrière + sortie barrière pour essayer de recouvrir utilement le temps de synchronisation.



Contrairement au SIMD il faut 2 conditions globales (3 états) :

1. repos
2. attendre que tous les PEs sont dans la barrière
3. attendre que tous les PEs ont pris en compte la barrière.

En SIMD : PEs synchrones \rightsquigarrow on sait si les PEs sont dans la barrière ou pas : une seule condition globale suffit.



SPMD ? *Single Programm Multiple Data stream ?* Modèle de programmation...

Multifibre (*multithread*)

- Mémoire trop lente multiplexage du processeur sur plusieurs fibres mémoire ↵ TERA MTA, 128 fibres
- Intrinsèque multifibre : processeur d'E/S du CDC6600, 10 PEs virtuels [Thornton, 1964]
- Processeurs trop lents : multiplexage de la mémoire sur plusieurs entités de calcul et d'E/S : Gamma 60 [Bull, 1957]



Idée : remplacer une boucle « vectorielle » par une opération vectorielle :

- ▶ Vectoriel si pas de problème de dépendances ↗ pipeline simple et rapide
- ▶ 1 instruction \Leftarrow : boucle scalaire : pas de goulet d'instructions (flot d'instruction faible)
- ▶ Pas de risque de dépendance de contrôle
- ▶ Nécessite une *if-conversion* pour le contrôle de flot parallèle
- ▶ Éléments de vecteur également espacés : mémoire banquée efficace (si pas de conflits sur les bancs). Compense l'absence de caches.



- De près : pipeline, plusieurs instructions à suivre sur un même flot de données (vecteur) \rightsquigarrow MISD ?
- Néanmoins : de loin, une instruction vectorielles manipule de nombreuses données \rightsquigarrow SIMD !



- Mémoire/mémoire : premières machines.
Simples car peu de limitation sur la taille des vecteurs
- registres/registres : machines modernes
 - ▶ localité pour appliquer plusieurs opérations
 - ▶ chaînage des unités vectorielles : relier ensemble les opérateurs
 - ▶ nécessite de découper les vecteurs en registres vectoriels : *strip-mining* ↪ performances maximales si nombre juste de registres vectoriels.



Qu'y mettre ?

Architecture flot de données...

- débit mémoire données très réduit : un seul flot traverse tous les opérateurs
- débit d'instruction important mais généralement programme simple (traitement du signal) ↪ mémoire de programme dans les PES
- style de programmation très particulier



- nécessite un problème découpable en tâches successives
- probablement plus restrictif que le SIMD.

Architectures systoliques.

iWARP : processeur MIMD traditionnel ! Seul le modèle de programmation est particulier...



Single Programme Multiple Data : plus un modèle de programmation qu'un type d'architecture.

En général :

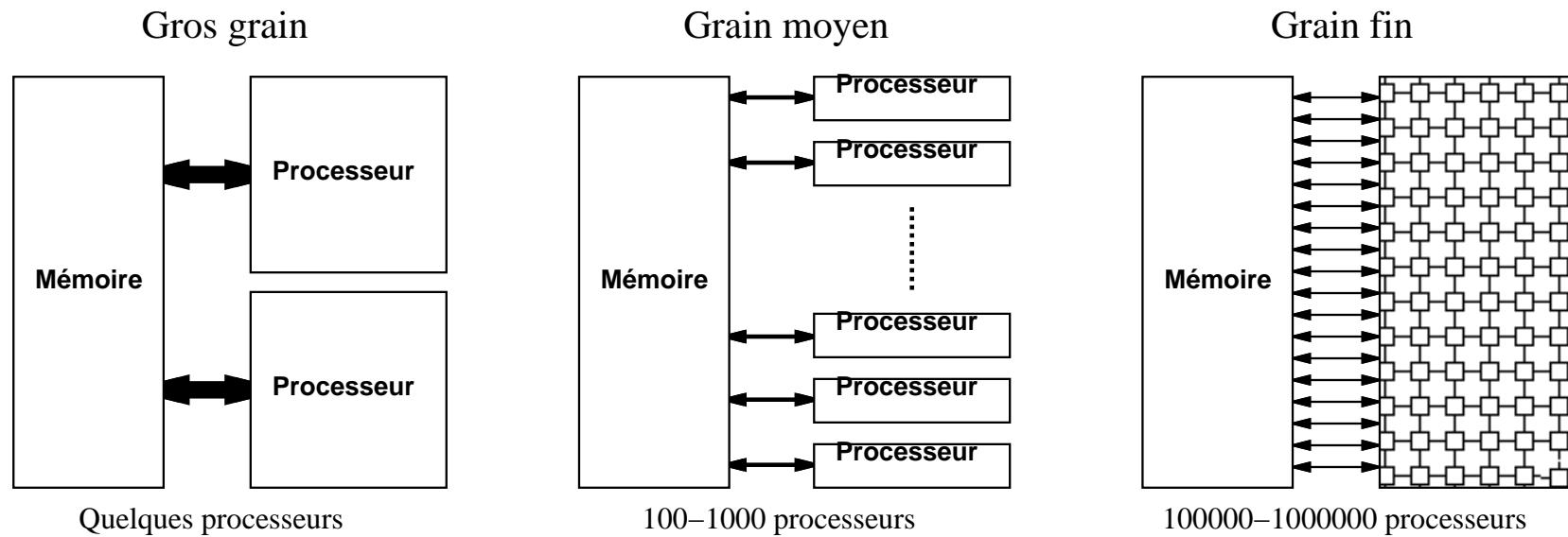
- MIMD
- système de synchronisation efficace.

Selon les machines on sépare ou pas la notion de processeur scalaire (pour les entrées-sorties).

CM-5 par exemple : PEs MIMD/vectoriels + nœud de contrôle + nœud d'entrées/sorties.



Parallélisme massif ? Compromis à trouver...



⚠ Pas de relation simple entre grain, nombre de processeurs et efficacité. Parallélisme massif en terme de « bits efficaces »...

Un gros grain exploite déjà du parallélisme en interne



Partir de l'ordinateur le plus puissance et en mettre un petit nombre (prix !) :

- toutes les machines vectorielles actuelles CRAY X-MP
[Chen, 1983] NEC SX-3R [NEC, 1991] CRAY Y-MP c90
[Cray, 1991]
- assez difficile de brancher entre elles ces bêtes de course
- souvent approche « une application par processeur »
- technologie chère



Faire les plus petits processeurs pour en avoir le maximum en supposant que $P = Np$:

- toutes les machines SIMD actuelles (CM-2 [TMC, 1987] MP-1 [Blank, 1990]) sauf MP-2
- ou les plus anciennes : [Unger, 1958] [Slotnick et al., 1962]
- processeurs plus simples à faire : machines commerciales à grain fin = projets universitaires !



Problèmes :

- 1 multiplications 32×32 bits = 1024 cycles sur PE 1 bit
- peu adapté aux formats de données classiques
- si efficacité en $\mathcal{O}(\frac{\infty}{\log N})\dots$
- plus de PEs \rightsquigarrow plus d'échanges \rightsquigarrow plus lent
- adressage de la mémoire locale difficile
- à réserver aux applications très spécifiques (traitement d'image 1 bit)

Exemple de la CM-2 : rajout de 2048 coprocesseurs flottants aux 64K PEs 1 bit

\rightsquigarrow les PEs 1 bit ne sont plus utilisés car ils *ralentissent* la machine !



Entre les 2 :

- exploitation des microprocesseurs de taille « classique »
- multiprocesseurs à mémoire partagée : BULL DPS 7000 [Bull, 1992], CAMPUS/800 [Alliant, 1991]
- MIMD à mémoire distribuée : PARAGON [Intel, 1991], CM-5 [:CM5], SPARC,...
- machines à base de TRANSPUTER [INMOS, 1991]
- profite pleinement du marché des processeurs :
 - ↗ performances. Débat obsolète si utilisation de processeurs standard...
- Machine ASCI à 9000 Pentium Pro, 1+ TFLOPS

Probablement l'avenir



- Évolution vitesse μ processeurs plus rapide que mémoire
 - Nombre limité (500) de « pattes » d'entrées-sorties
- ~~~ Limitation des performances par la mémoire
- ▶ Intégration mémoire et processeur ? Technologie encore incompatible entre logique et mémoire impliquerait trop faible capacité mémoire
 - ▶ Hiérarchisation de la mémoire à bande passante constante : mémoire cache pour diminuer temps d'accès
 - ▶ Parallélisation de la mémoire : mots larges (32, 64, 128, 256, 512 bits... Nombre de fils !) & mémoire banquée pour augmenter le débit



Localité dans les programmes :

Temporelle : une donnée accédée risque de l'être prochainement

Spatiale : une donnée accédée risque d'être suivie par un accès voisin

Encore plus vrai dans du code (boucles, fonctions, fichiers,...)



~~ Instauration d'une hiérarchie basée sur le compromis taille × vitesse :

1. Registres du PE (1 ns) & registres vectoriels
2. Tampons d'instructions
3. Mémoire cache primaire (10 ns), secondaire, tertiaire,...
4. Mémoire principale (100 ns)
5. Mémoire étendue (CRAY)
6. Disque magnétiques (10 ms)
7. Disques optiques
8. Bandes (migration) (minutes)



Mémoire actuelle poussée désormais par

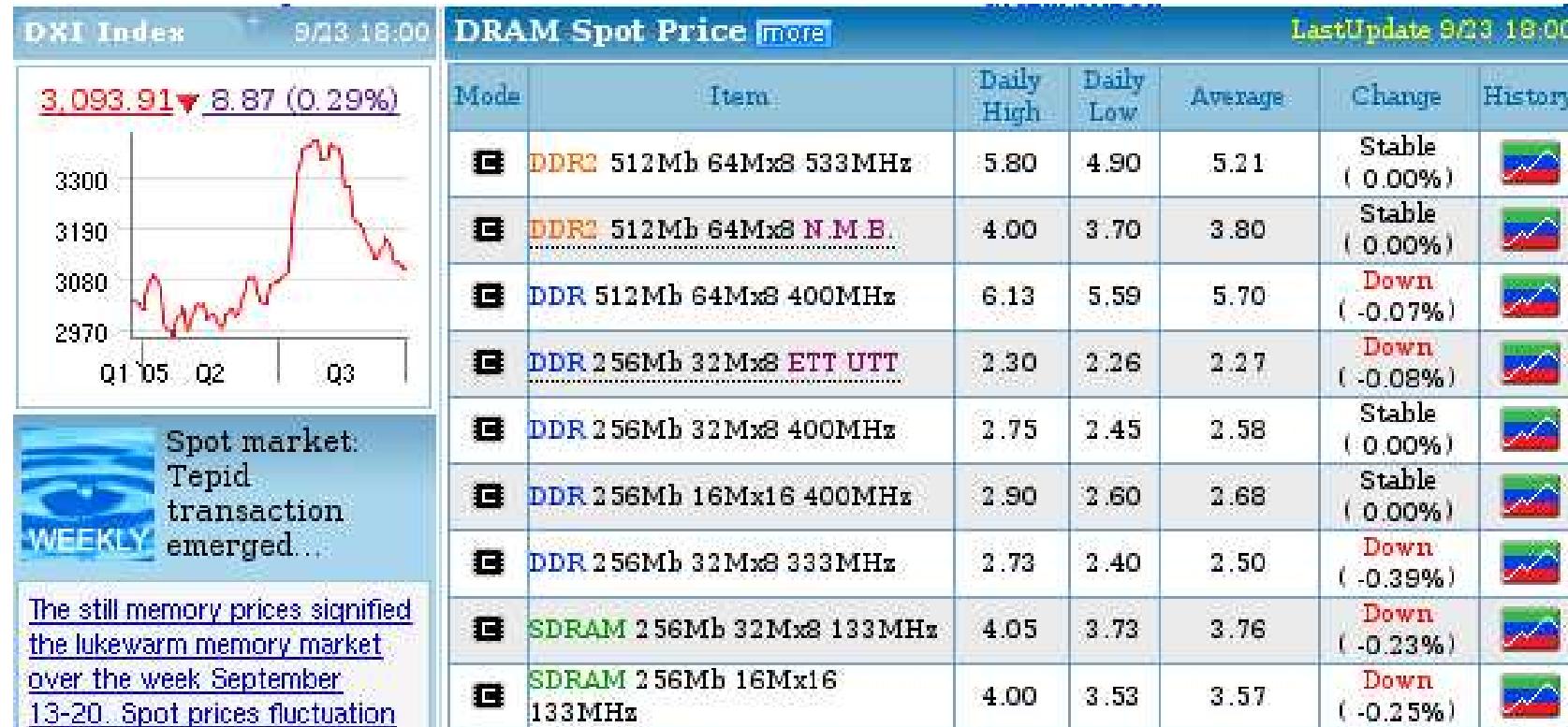
- Marché de masse
- Joueurs extrêmes
- Applications multimédia

qui sont maintenant « Haute Performance »

<http://www.simmtester.com>



<http://www.dramexchange.com>



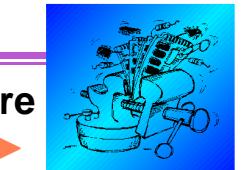
Il existe des DRAM UTT (UnTesTed !) un peu moins chères...

<http://www.simmtester.com/page/news/showpubnews.asp?num=124>

- Permet de rentabiliser les wafers foireux



- Externalise le test à d'autres sociétés
- Recyclable dans des d'autres applications faible performance
(baladeurs MP3, répondeurs téléphoniques, lecteurs DVD,...)



- Organisation depuis 1975 en tableau de bit avec commande selon une ligne (rangée) puis une colonne
- Adresse envoyée d'abord suivant la rangée (échantillonnée sur RAS (*Row Address Strobe*) puis la colonne (échantillonnée sur CAS (*Column Address Strobe*))
- Stockage dans un simple condensateur ↗ nécessité de « rafraîchir » la donnée régulièrement (moins de 5 % du temps)
- Par commodité, mémoires vendues souvent sous forme de barettes SIMM (*Single*) ou DIMM (*Dual Inline Memory Module*)
- Gain en vitesse :
 - ▶ Grand débit interne disponible (parallélisme sur les rangées)
 - ▶ Éviter un cycle de RAS si localité dans la même page : *Fast Page Mode* ↗ se contente de lire dans le tampon de sortie des rangées



- ▶ Évite des verrous externes pour échantillonner les signaux ↗
SDRAM (*Synchronous DRAM*) avec bascules D à l'intérieur.
Marque commerciale PC f où f est la fréquence d'horloge en
MHz (ex. PC133)
- ▶ Échantillonnage des signaux sur front montant et descendant
↗ DDR (*Double Data Rate*) 64 Mb–1 Gb, baisse de la tension
d'alimentation (2,5 V). Nom commercial PC d d'une barette
DIMM de 8 octets de large avec du DDR333 (167 MHz
d'horloge) : $d = 333 \times 8 = 2667$ ↗ PC2700
<http://download.micron.com/pdf/technotes/TN4605.pdf>
- ▶ DDR2 pour des transferts de plus de 400 MHz, 256 Mb–4 Gb,
diminution consommation
 - 1,8 V
 - Mode de terminaison des lignes programmable et
synchrone en multi-banc sur les écritures



- Taille de page (rangée) moitié par rapport au DDR : division consommation par 2 lors d'une commande ACTIVATE (\approx RAS)
- Réglage fin du pipeline des opérations
- DDR2-667 MHz (barettes PC2-5400) 5-5-5 (latence CAS (CL), latence RAS (RCD), latence précharge (RP) en cycles) a une latence de 5 cycles CAS : $3 \times 5 = 15$ ns
- Si accès aléatoire, latence totale = CL + RAS \rightsquigarrow 10 cycles, 30 ns sur premier bit d'une page
- Si accès aléatoires en permanence à la mémoire, temps de précharge en plus, 15 cycles, 45 ns
- Latence semblable à de la DDR mais débit double
45 ns pour Opteron 3 GHz = 1620 instructions !
 ☺<http://download.micron.com/pdf/technotes/ddr2/TN4702.pdf>



http://download.micron.com/pdf/datasheets/modules/ddr2/HTF16C64_128.pdf

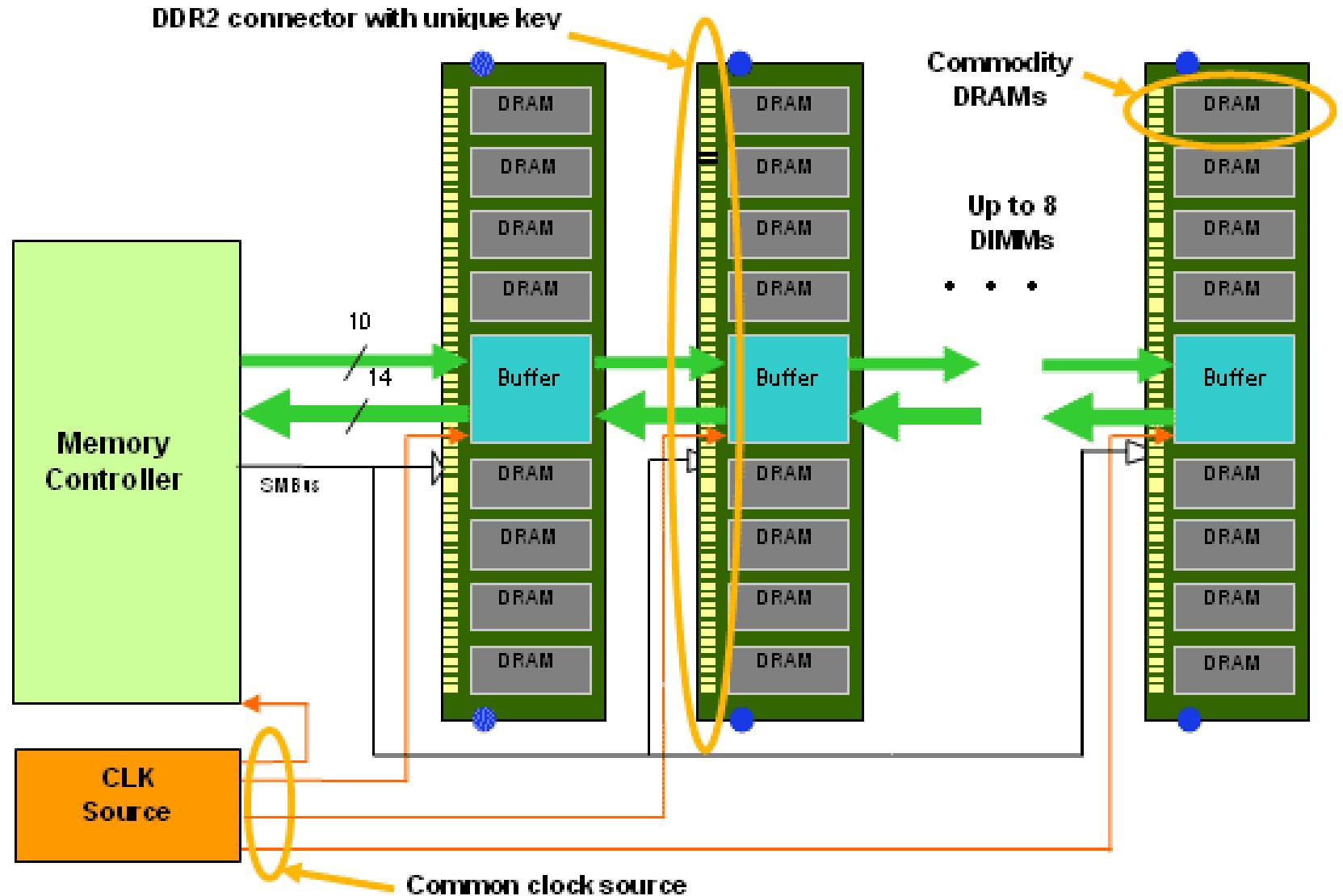
La localité est toujours importante !

- ▶ RAMBUS
 - Remplacer signaux mémoires DRAM classiques par des bus rapide à transactions éclatées (lecture/écriture, retour,...)
 - Interfaces plus chères
- ▶ Projets de recherche PIM (Processors In Memory) : énorme débit processeurs-mémoires si local
 - ~~> Revoir modèles de calcul/programmation ?



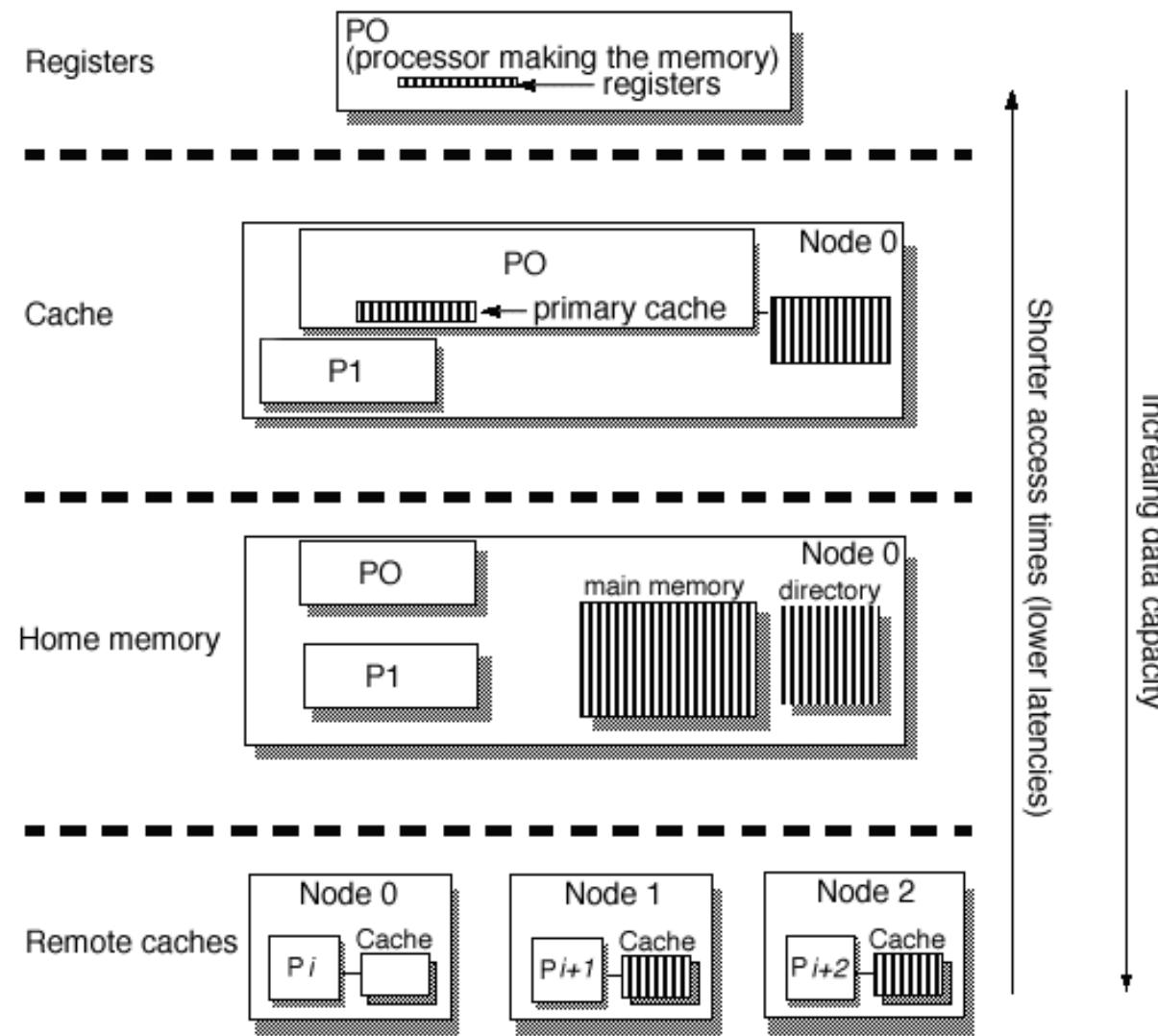
- FB-DIMM (Fully Bufferer DIMMs)
 - ▶ Rajoute un ampli sur chaque barrette pour limiter bruit et améliorer débit
 - ▶ Interface du DDR2/3 avec un bus série : simplifie les circuits imprimés (69 pattes au lieu de 240) ↗ racks 1U, *blade*
 - ▶ Gestion des erreurs de communication en plus de l'ECC
 - ▶ 192 Go, 6 canaux, 8 DIMMs/canal, 6,7 Go/s/canal ↗ 40,2 Go/s



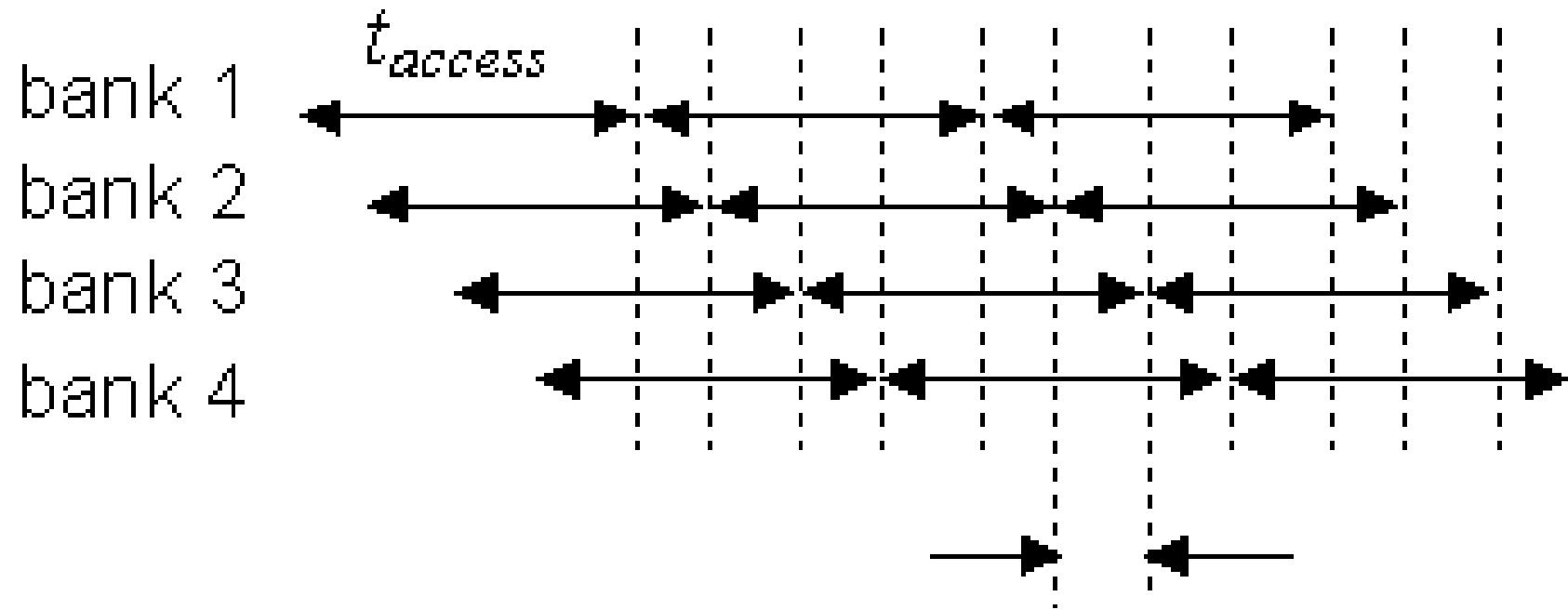


<http://www.simmtester.com/page/news/showpubnews.asp?num=113>





Cacher la latence en distribuant les requêtes sur différentes mémoires



$$t_{average} \approx \frac{t_{access}}{n}$$



- Nécessite des accès bien distribués ↗ entrelacement cyclique des adresses. Exemple à 4 bancs :
Banc 0 : adresses 0,4,8,12,...
Banc 1 : adresses 1,5,9,13,...
- ⚡ accès à des tableaux par pas $\neq 1$ (et non premier avec nombre de bancs) : risque de taper dans un sous-ensemble de bancs... Revoir les codes !
- Stations de travail haut de gamme, supercalculateurs (1024 bancs et 250 Go/s sur C90)



Rêve de l'utilisateur λ : ne voir qu'une grosse machine SISD
~~~ énorme mémoire globale.

Problèmes (à défaut de SISD...) :

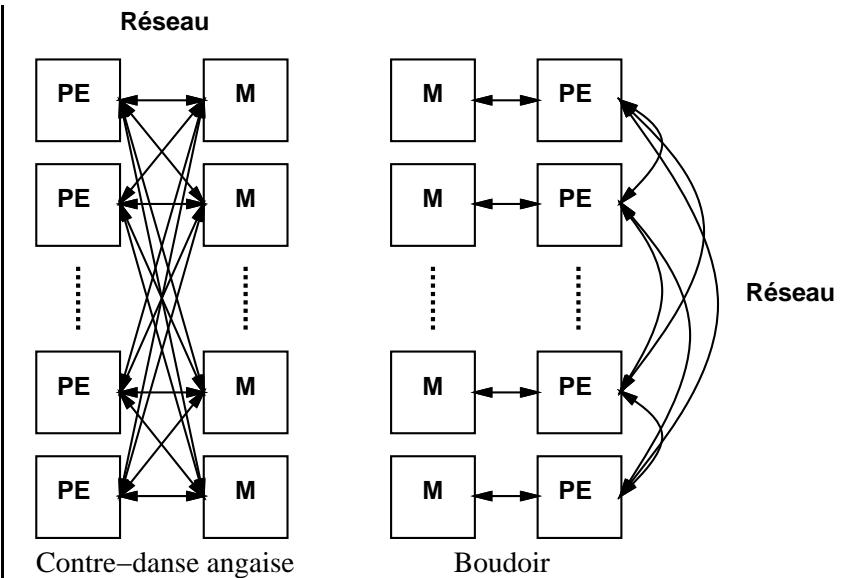
- Sortance des portes logiques bornées (temps en  $\mathcal{O}(\mathcal{M})$ ) =:  
amplifications exponentielles nécessaires (temps en  $\mathcal{O}(\log \mathcal{M})$ )
- Temps de propagation de l'ordre de  $\frac{2}{3}c$  dans les fils
- On peut augmenter le débit mais pas diminuer la latence
- Conflits d'accès à une même case par plusieurs PEs



Solutions possibles :

- Exploiter la localité des accès
- Couper la mémoire en morceau

Mais comment relier processeurs et mémoire(s) ?



**Couplage fort contre couplage faible.**



Avantages :

- Plus simple à programmer (mais non sans surprise...)
- Logiciel plus simple : chaque PE voit toutes les autres mémoires
- Découplage entre nombre de PEs et nombre de mémoires
  - ▶ Beaucoup de bancs ↗ fort débit
  - ▶ Nombres premiers pour éviter des conflits, etc.
- Mémoire maximale vue par *chaque* processeur, utile pour applications demandant beaucoup de mémoire



Inconvénients :

- Optimisation du cas pire : « on ne sait rien et on fait le maximum »
- Matériel complexe car débit élevé en continu
- Cohérence mémoire délicate si cache
- Difficile si beaucoup de PEs

Mémoire souvent réalisée de manière structurée : bancs.

Pour grosses machines vectorielles ou les multiprocesseurs (mainframes ou stations de travail), BSP, à faible nombre de processeurs : CRAY Y-MP C916 : 1024 bancs pour 250 Go/s [Cray, 1991]...



Avantages :

- Optimisation du meilleur cas : « on sait tout et on fait le minimum », en ayant tout localement
- Rapide si placement des données correct  $\rightsquigarrow \neq$  localité
- « Compilation » des communications : cache logiciel, fusion de messages
- Assez simple même si beaucoup de PEs



Inconvénients :

- Si accès très dispersés : lent
- Placement (distribution et alignement) important
- Temporaires pour stocker les données communiquées
- Nécessite un logiciel plus complexe.

Toutes machines SIMD sauf BSP : CM-2, MP-1, ILLIAC IV, OPSILA.

Machines MIMD avec beaucoup de PEs : ASCI Intel & IBM, CM-5, TRANSPUTERS.

Conditionne la programmation...



Éviter les conflits sur des accès typiques (lignes, colonnes, diagonales si possible) :

- $(i,j) \longrightarrow (ui + vj) \bmod N$  BSP [Lawrie, 1975]  
[Lawrie and Vora, 1982]
- $(i,j) \longrightarrow i \oplus j \bmod N$  STARAN
- $(i,j) \longrightarrow AI \oplus BJ \bmod N$  XOR scheme [Frailong et al., 1985]
- carrés magiques  $N \times N$  avec des nombres de 1 à  $N$   
[Balakrishnan et al., 1988]

Problème : temps de calcul ↗ faire simple...

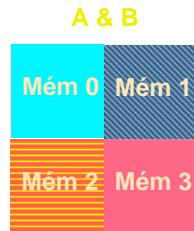


Multiplication de matrice :

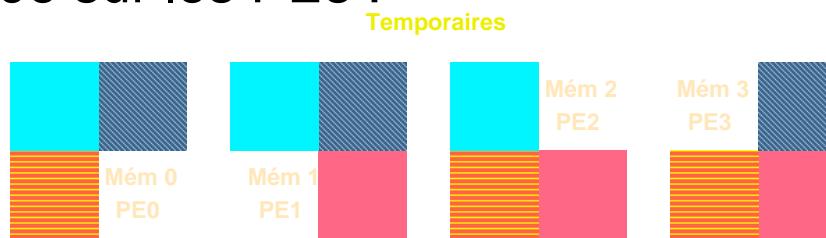
$$B = A \times A$$

## Modèle d'exécution et couplage mémoire

- ▶ Mémoire partagée :



- ▶ Mémoire distribuée sur les PEs :



Communication dans 1 temporaire (voire 2...) nécessaire pour des calculs répartis aussi par blocs



MP : Multi-Processeur... à mémoire partagée  $((MP)^2)$

Au niveau système :

## SMP

- ▶ : symétrique

Chaque processeur peut accéder au matériel d'E/S  
(sémaphores...)

## AMP

- ▶ : asymétrique

Seul 1 processeur accède au matériel. Les autres sous-traitent  
SMP plus compliqué (SunOS 5) mais plus efficace que l'AMP  
(SunOS 4) pour les E/S.



Relier entre eux PES-PEs ou PEs-mémoire(s) :

**LE facteur clé limitant des machines parallèles :**

- ▶ les PEs sont de plus en plus rapides
  - ▶ les programmes sont de plus en plus gourmands
  - ▶ coût important : tendance à utiliser la marge de « GROSCH modifiée »
  - ▶ protocole standard ? Inutile, réseau spécifique.
- 
- Limiter le nombre de fils
  - Préférer les réseaux « réguliers »

Équilibre entre coût et performance  $\equiv$  le pari d'une machine !



Étude du graphe sous-jacent d'un réseau :

**sommet** : commutateur ou PE

grosse machine  $\Rightarrow$  #sommets

**arc** : lien de communication

forts débits  $\rightsquigarrow$  liens larges et rapides

**diamètre** : nombre de liens

faible latence  $\rightsquigarrow$  diamètre

**degré** : nombre de liens par PE

débit  $\rightsquigarrow$  degré  $\rightsquigarrow$  filasse !





Réalisation = projeter le tout dans l'espace *physique* 2D ou 3D !

Connaissance topologique :

- ▶ idée théorique des performances et de la réalisabilité.
- ▶ information théorique pour le routage : comment faire cheminer au mieux l'information ?

Topologie temporelle :

**figée** : réseau statique

pas de latence de configuration

**évolutive** : réseau dynamique

adaptable aux besoins de l'algorithme mais configuration lente.

Réseau des TRANSPUTERS T800.



## centralisé : optimisations globales

- évite des conflits
- nécessite un réseau de contrôle sous-jacent (qui peut avoir des conflits...)
- goulet d'étranglement sur le contrôleur

## local : parallélisation du contrôle

- vision locale ↪ non optimal
- décisions rapides.

Avenir au contrôle local mais certains réseaux nécessitent un contrôle global.



**Commutation par paquets** : données envoyées par paquets qui se propagent de proche en proche

**Commutation par circuits** : ouverture d'un canal de communication réservé dans le réseau. Permet d'envoyer directement des données entrée → sortie.

Systèmes intermédiaires pour récupérer les avantages des 2.

Virtualisation d'une notion par rapport à l'autre possible.



Opérations précédentes synchrones ou asynchrones :

- synchronisme plus simple si contrôle global
- dans les circuits le synchronisme est actuellement plus simple
- sur des grands réseau l'asynchronisme est plus simple (propagation)
- synchronisme difficile à grande vitesse entre les circuits.

Choix : problèmes fins de réalisation.



Étudier le produit cartésien des ensembles précédents.

En fait, ensembles pas totalement orthogonaux :

- commutation et staticité : un réseau dynamique = réseau statique dont certains sont des commutateurs
- possibilité de réseaux hybrides ou hiérarchiques.

Choix en fonction des besoins [Wu and yun Feng, 1984].



Nombreux paramètres et critères souvent contradictoires :

**degré** : • ↓ filasse et pattes de circuit ↼ ↓ degré

• ↗ débit ↼ ↗ degré

• régularité des nœuds ↼ degré constant

**diamètre** : ↓ latency ↼ ↓ diamètre

**routage** : possible (!) et simple, si possible local et adaptatif

**symétrie** : évite des embouteillages, de particulariser nœuds et routage

**débit** : important pour ne pas trop brider les PES

**coût de démarrage** : faible pour diminuer la latence



**largeur des liens :** • ↗ débit ↵ largeur

• ↘ pattes de circuits ↵ ↘ largeur

**performance :** adéquation avec les applications visées

**extensibilité :** éviter que nombre de liens  $\rightarrow \infty$  avec  $N$

**tolérance aux pannes :** nécessite plusieurs chemins

**réalisabilité :** dans l'espace euclidien

**partitionabilité :** découpage en baies, racks, cartes, circuits.

↔ trouver **LA** bonne fonction de coût. Par exemple

- nombres de pattes des circuits
- bisection du réseau
- nombre de liens.

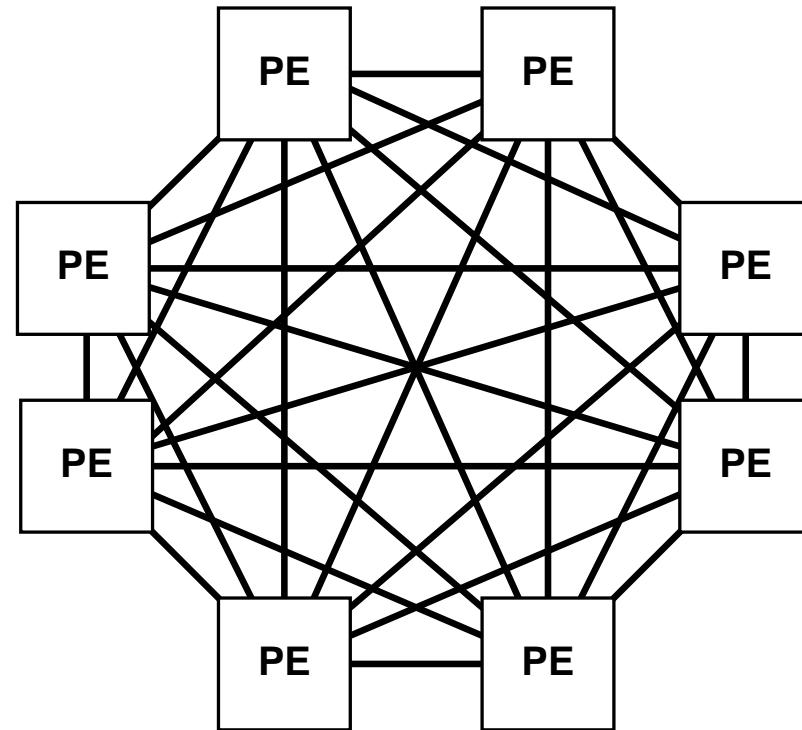


- PES aux sommets d'un graphe à topologie fixe
- 1 PE ne peut communiquer qu'avec ses « voisins »
- Communications distantes : demander à son voisin de faire suivre le message
- Optimisation des communications entre voisins
- Utile pour des algorithmes réguliers (traitement d'image, calcul systolique)



Relier chaque PE avec tous les autres :

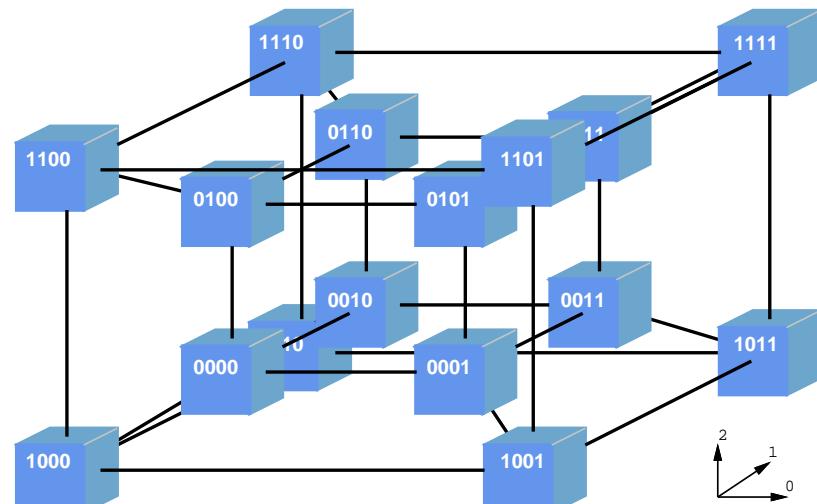
- ▶ Coûteux :  $N(N - 1)/2$  liaisons
- ▶ Trop performant : peu probable qu'un PE puisse soutenir en même temps toutes ses communications.



Généralisation du carré et du cube :

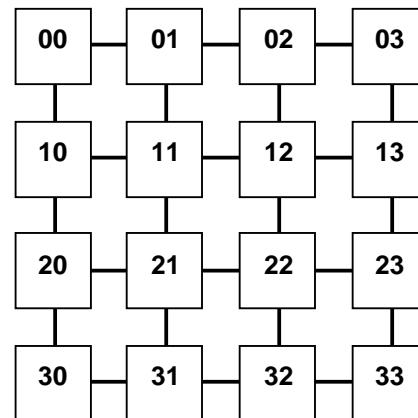
- ▶ Diamètre assez faible  
 $n = \log_2 N$
- ▶ Plusieurs chemins optimaux (combien ?)
- ▶ Construction récursive
- ▶ Si  $\nearrow N$  assez difficile à

construire,  $\nearrow n$  et espace 3D

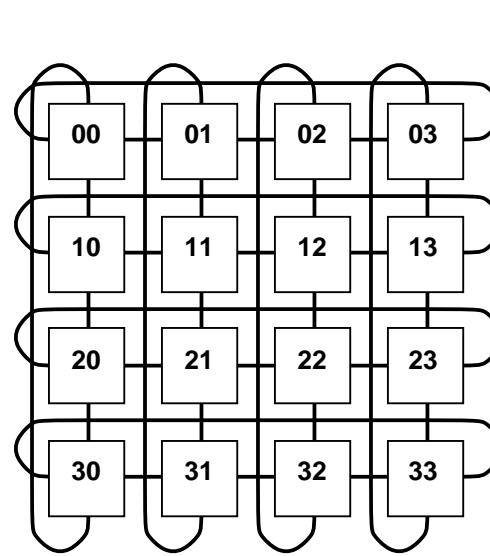


Remplacer chaque arrête de l'hypercube par une ligne de  $k$  processeurs

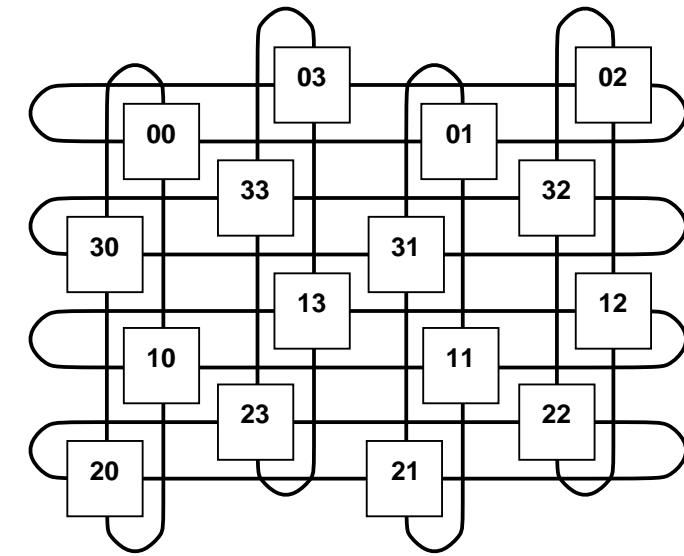
- ▶ Diamètre  $nk - 1$ , moyen  $\frac{nk}{2}$
- ▶ Tore si rebouclage sur les bords
- ▶ Avantage en 2D et 3D : utilise pleinement l'espace physique (bisection en  $\mathcal{O}(\mathcal{N}^{\frac{2}{3}})$ )



Hypergrille



Hypertore



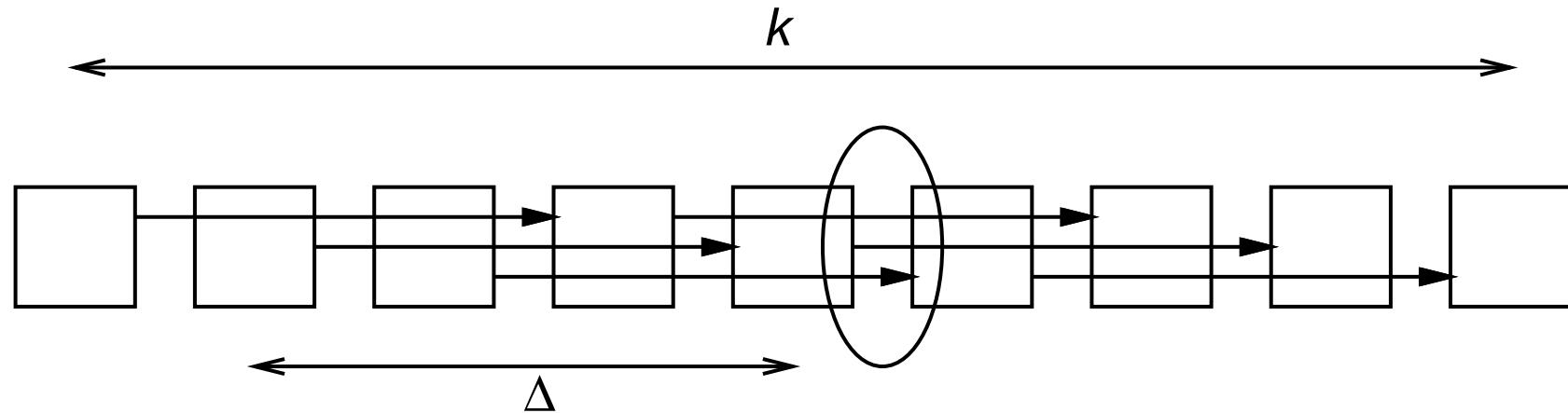
Hypertore



Voie en vogue : INTEL ASCI (grille 2D), CRAY T3E (grille 3D)



Paradoxe : cas des communications sur grilles distantes :



Conflit  $\rightsquigarrow$  temps  $\mathcal{O}(\|\cdot\|)$ . Séquentialisation  $\rightsquigarrow \mathcal{O}(\cdot^\epsilon)$

Comparaison entre un hypercube et une grille pour  $N = 256$  :

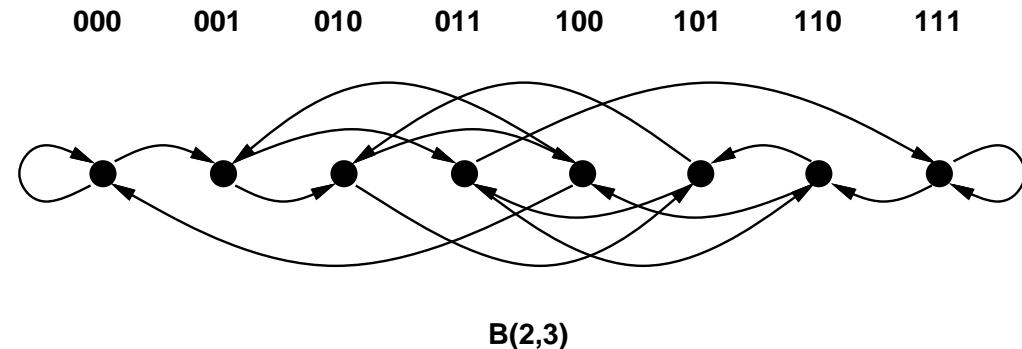
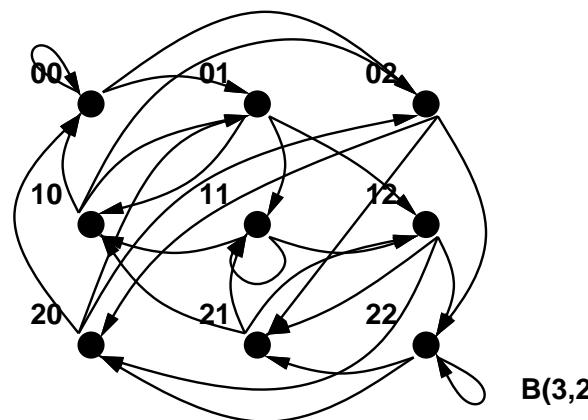
- ▶ hypercube de dimension 8, diamètre moyen 4
- ▶ tore 3D  $8 \times 8 \times 4$ , diamètre moyen 5
- ▶ grille 3D, diamètre moyen 10, compensable par liens larges



Diamètre/lien très proche de l'optimalité (critère de MOORE).

DE BRUIJN  $B(d,D)$  :

- ▶ nommer chaque sommet avec un mot de  $D$  lettres dans un alphabet de  $d$  lettres  $\rightsquigarrow d^D$  sommets
- ▶ un sommet  $(x_1, x_2, \dots, x_D)$  est relié à tout sommet  $(x_2, \dots, x_D, \alpha)$ ,  $\alpha$  lettre de l'alphabet



- ▶ algorithme de routage en  $D$  cycle (non optimal) : cycle de routage = décalage d'une lettre et placement d'une lettre de la destination pour choisir le bon lien
- ▶ réseau bidirectionnel non intéressant car diamètre  $D$
- ▶  $B(2,D) = \text{shuffle exchange}$  basé sur le « battage parfait » (algorithmes de FFT)

Extension : empêcher 2 lettres consécutives dans les noms ↗  
graphe KAUTZ, plus de boucle.

Problème ouvert : émulation de grilles dans ces réseaux.

Construction difficile.

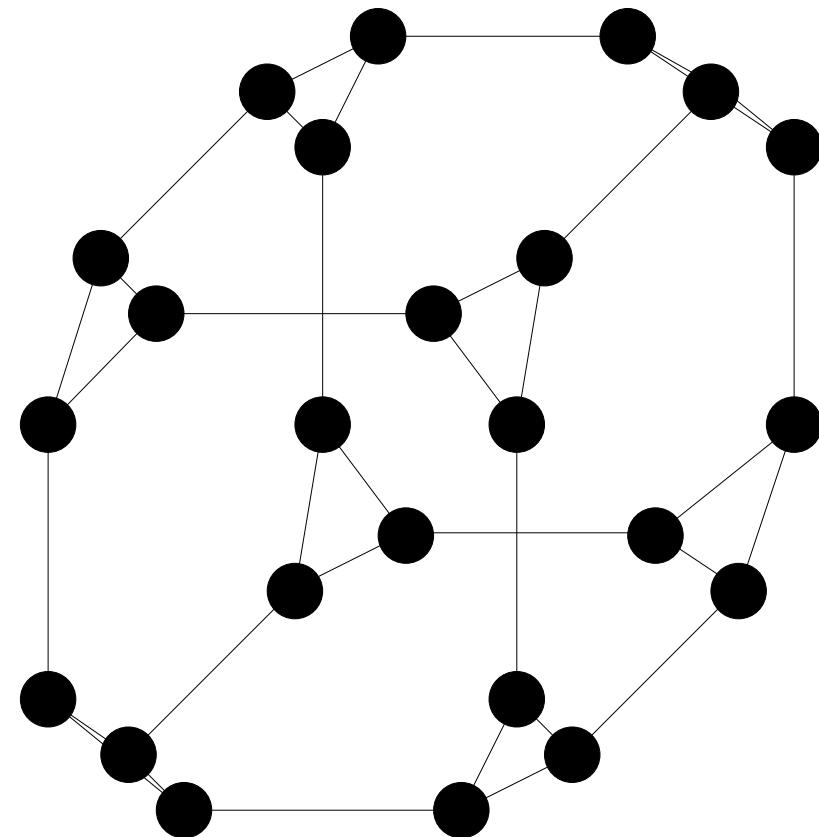


Remplacer les sommets d'un hypercube par des anneaux  
[Preparata and Vuillemin, 1981] :

- degré 3 indépendant de  $n$
- $n2^n$  sommets
- extensible

Routage non optimal si simple

Type d'extension pouvant être appliquée à d'autres graphes



## star-graph : [Akers et al., 1987]

- nom de sommet à  $n$  lettres différentes choisies dans un alphabet de  $n$  lettres
- $n!$  sommets
- un sommet est connecté à ceux dont le nom est identique à l'échange de la première lettre avec une autre
- degré  $n - 1$
- diamètre raisonnable  $\lfloor \frac{3(n-1)}{2} \rfloor$
- difficile à représenter et à construire...



**graphes aléatoires** : routage (local)...

**graphes de CAYLEY** : [Campbell et al., 1992]

- prendre un groupe
- choisir (aléatoirement) un sous-ensemble générateur
- le passage d'un sommet à un autre se fait en appliquant un élément du sous-ensemble

Problème du routage dans les graphes exotiques



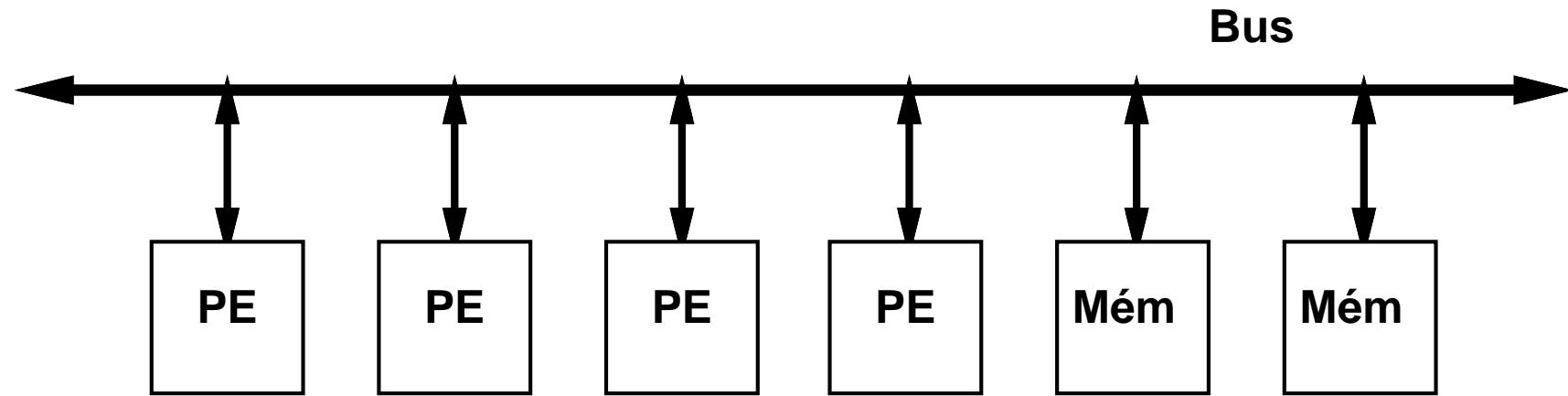
Commutation de circuits : utilisé avant l'informatique dans les centraux téléphoniques

- ▶ Rapide une fois que la connexion est établie
- ▶ Chemin occupé même s'il n'y a pas de données
- ~~> Compromis entre nombre de chemins établis, nombre de commutateurs, type de commutation

Commutation : déroulage spatial de l'algorithme de routage du réseau statique ~~ compromis espace-temps

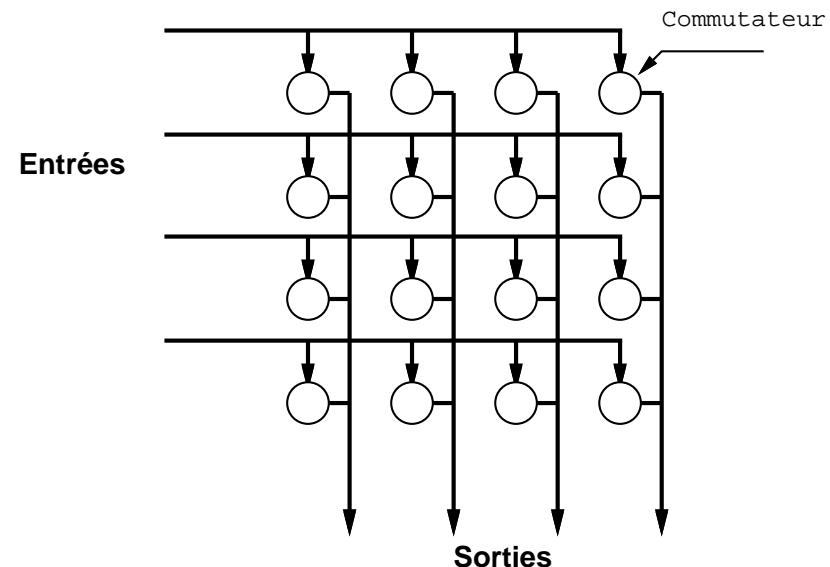


- Tout sur un même support
- Version à commutation de paquets : bus à séparation des requêtes et des réponses
- Matériel simple  $\mathcal{O}(N)$  mais bande passante limitée



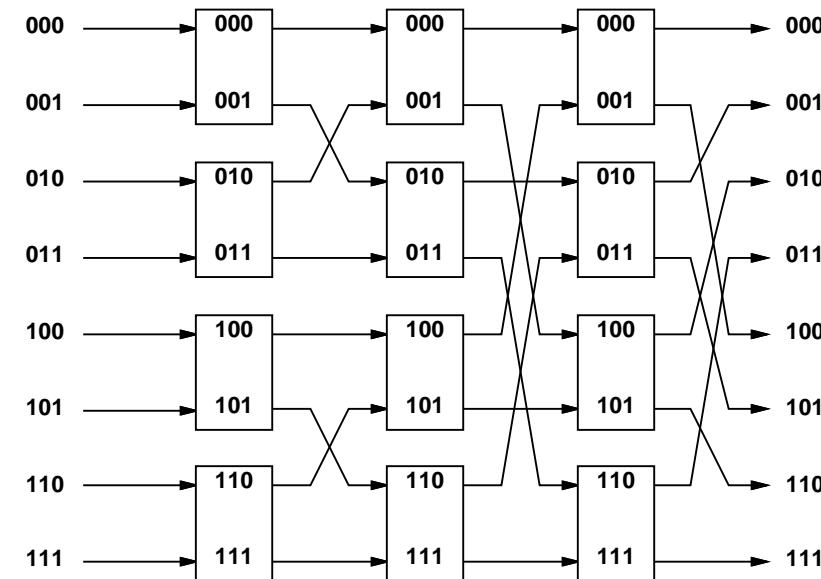
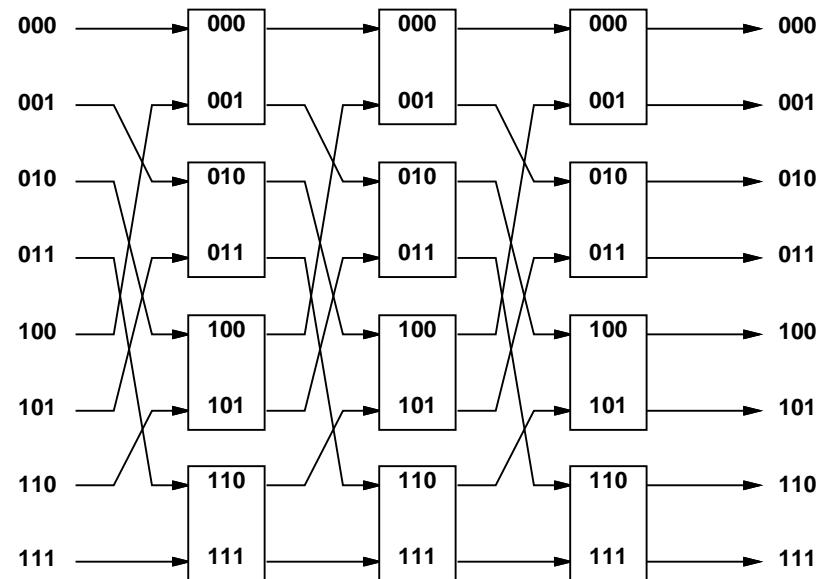
*Crossbar ou matrice de croisillons :*

- ▶ entrée → sortie en 1 commutateur
- ▶  $N^2$  commutateurs
- ▶ contrôle très simple, très rapide



Partir d'un réseau  $B(2,D)$  et dérouler le réseau  $D$  fois pour le routage ↗ réseau multiétage [Lawrie, 1975]

- ▶  $N \log_2 N$  commutateurs
- ▶ moins de chemins que le crossbar =: plus bloquant



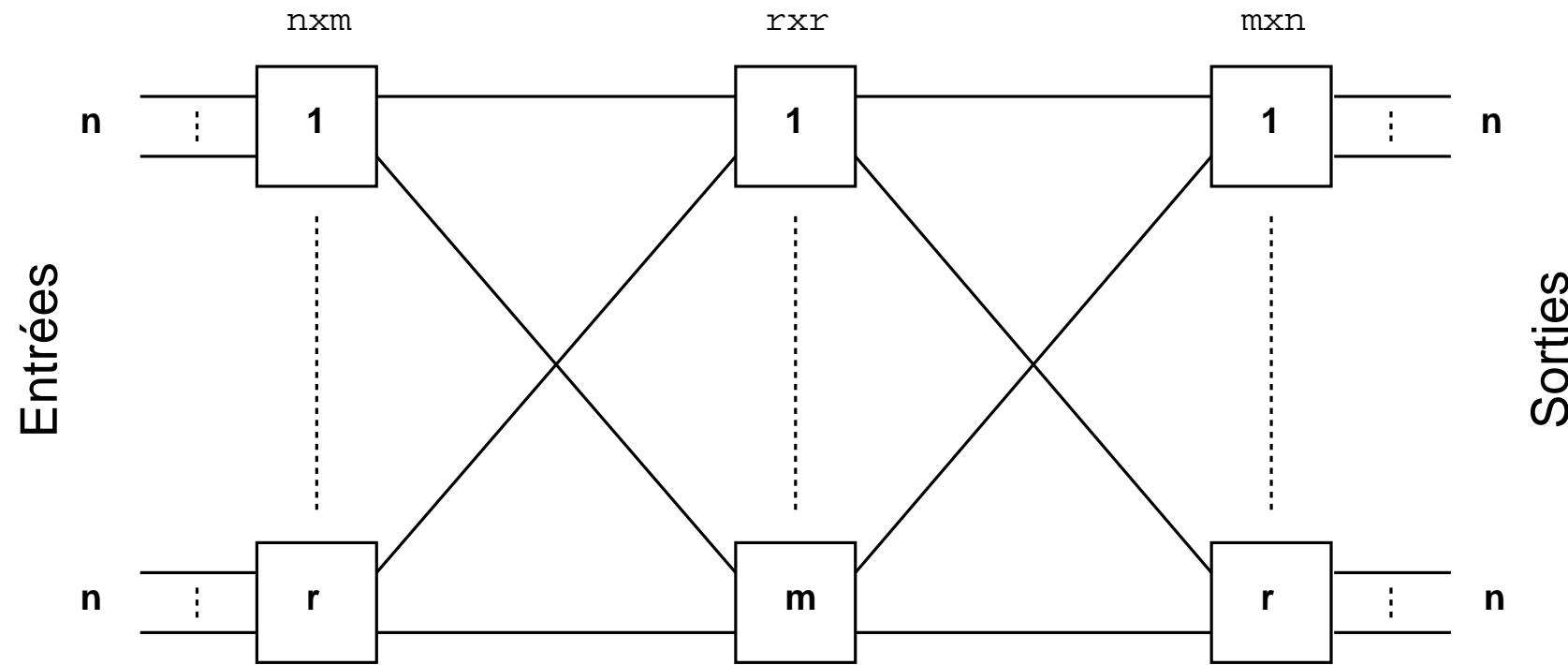
- ▶ routage simple : *destination tag algorithm* (DTA). À chaque étage le bit d'adresse correspondant sélectionne le commutateur
- ▶ nombreux réseaux (isomorphes depuis [Wu and Feng, 1980] [Bermond and Fourneau, 1988]...) : *indirect binary  $n$ -cube* FLIP de STARAN (Oméga inversé), Delta,... Différents pour le câblage et le partitionnement
- ▶ généralisation avec  $B(k, \log_k N)$  :  $k \log_k N$  commutateurs



Téléphone : pouvoir rajouter une communication sans couper aucune communication, sans réarranger le réseau [Clos, 1953]

- trois étages de crossbars
- non bloquant (puissance du crossbar) si  $m \geq 2n - 1$
- $Nm(2 + \frac{1}{n})$  points de croisement, soit au moins  $N^2(\frac{4}{r} + \frac{2}{rn}) + N(\frac{1}{n} - 2)$
- récurrence possible
- problème : contrôle non trivial



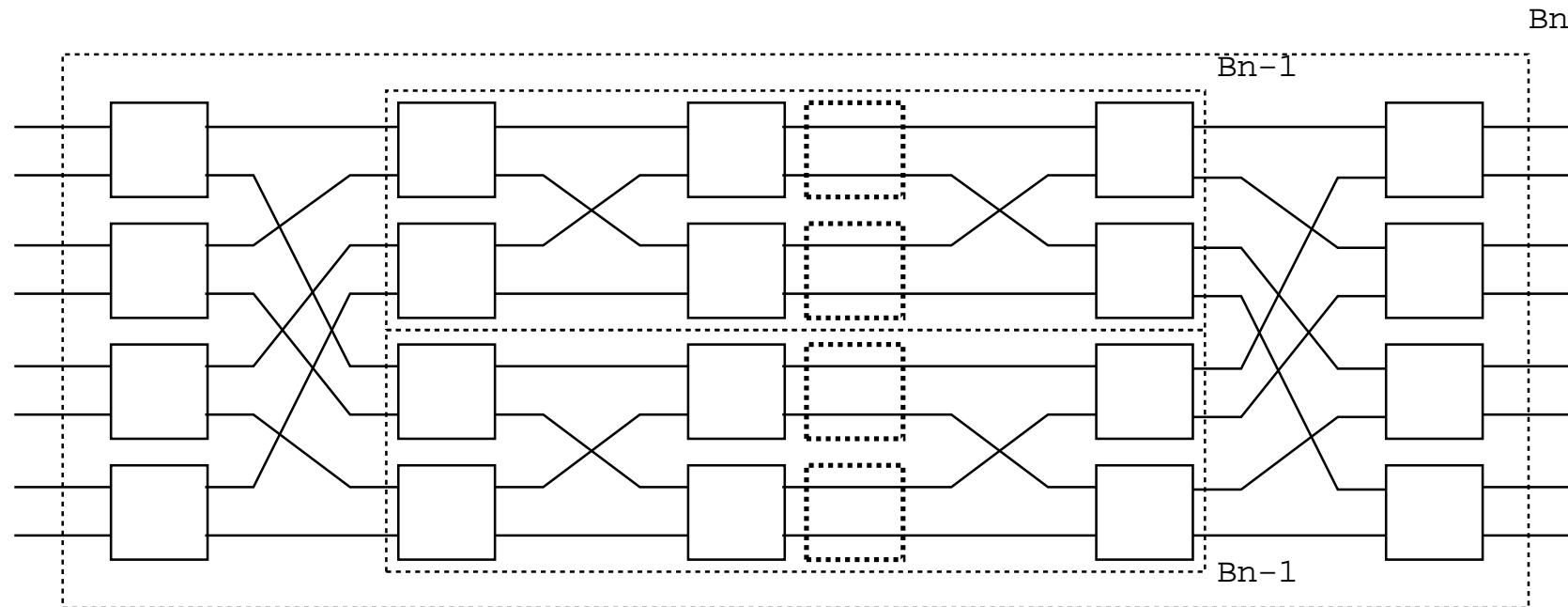


Simplification du réseau de CLOS si  $n$  et  $r \times n$  puissance de 2,  $m = 2$

[Beneš, 1962] :

- construction récursive
- un réseau Oméga + un réseau FLIP
- $\frac{N}{2}(2 \log_2 N - 1)$  commutateurs
- non respect de  $m \geq 2n - 1$  : réseau réarrangeable
- pas de routage rapide connu





- *data manipulating functions* (DMF) [Feng, 1974] basé sur une série de décalage. Redondance de chemins
- construit à partir d'un hypercube [Szymanski, 1989]
- réseau de tri [Batcher, 1968] autoroutant avec  $\frac{N}{4} \log^3 N$  commutateurs
- hybride BENEŠ-BATCHER : l'autoroutage avec le non bloquant et l'économie [Koppelman and Oruç, 1990].



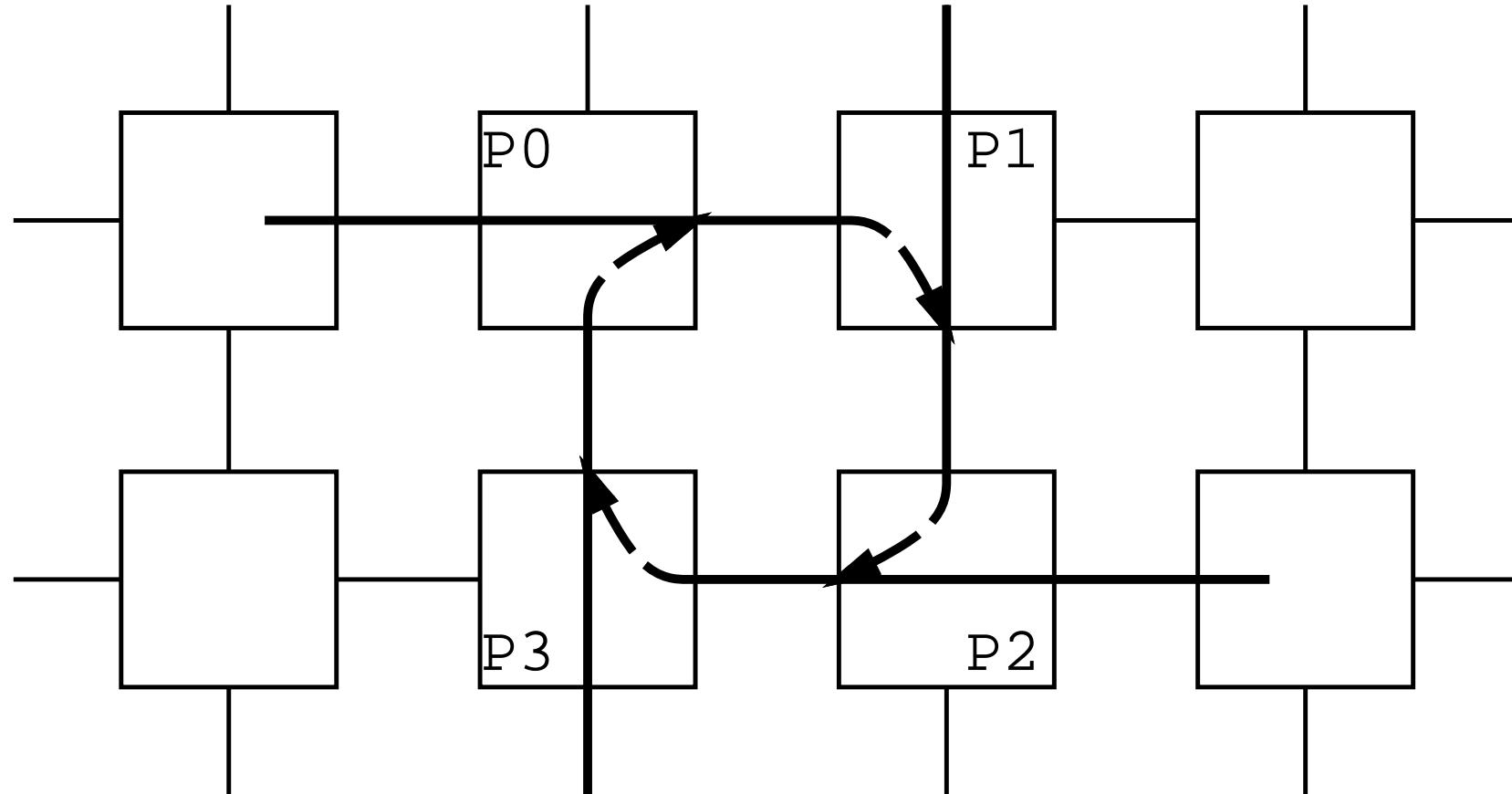
- paquets : chaque nœud reçoit un paquet et le fait suivre
- circuits : commencer par établir tout le circuit avant d'envoyer un paquet
- *virtual cut-through* : pipeline du message  
La tête établit le circuit qui se défait derrière le message Si bloquage, message stocké dans le commutateur bloquant
- *wormhole* : idem sauf simplification : le message reste bloqué dans le réseau *in situ*. Machines SYMULT 2010, nCUBE-2, iWARP et INTEL PARAGON



Commutation de circuit : les conflits sont gérés à l'ouverture

Commutation de paquets : conflits générés en cours de propagation ↵  
étreintes mortelles possibles si cycle dans les ressources  
nécessaires !



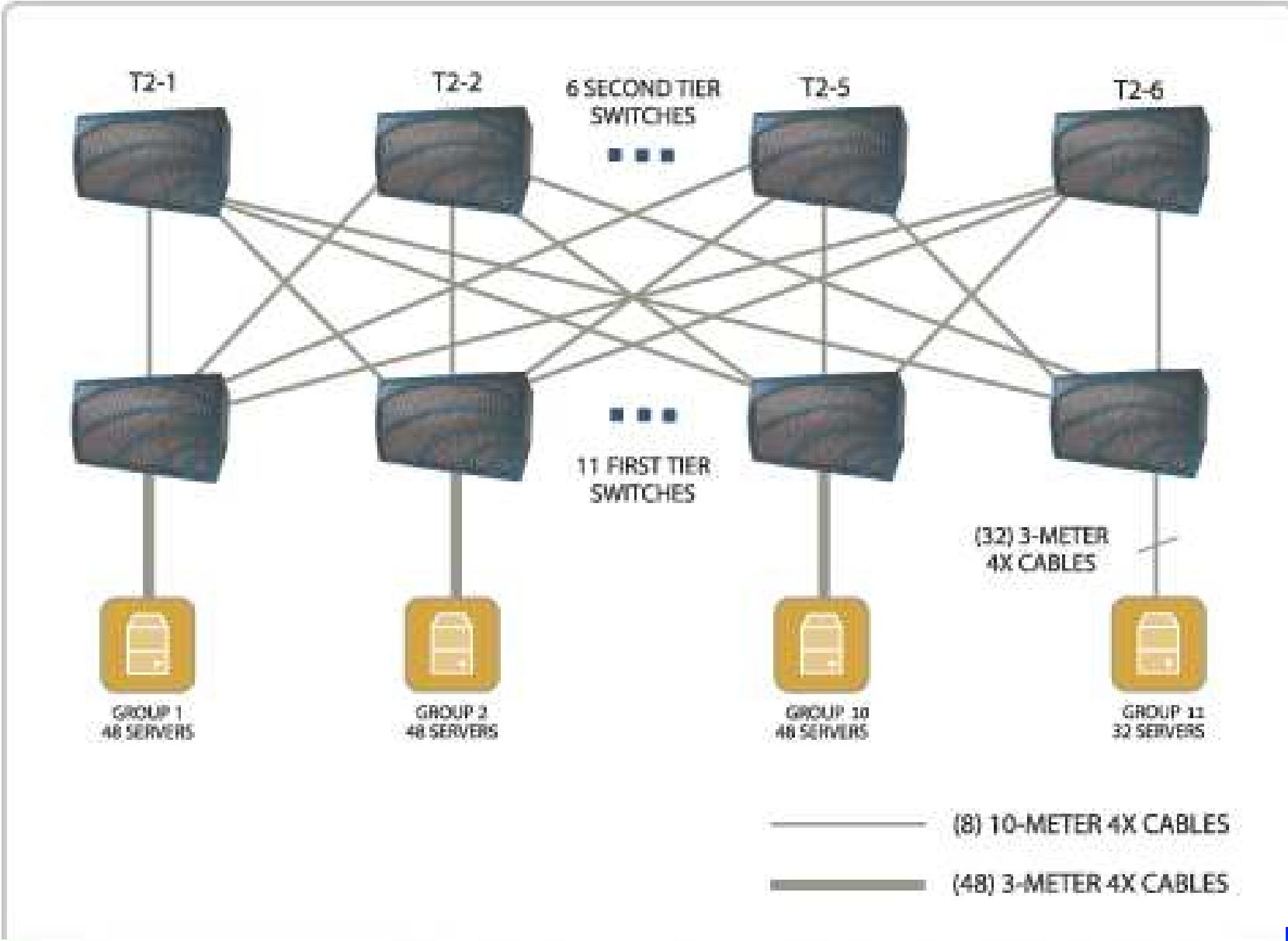


- Algorithme déterministe : obliger les paquets à utiliser un chemin sans cycle de dépendance  
Ex. sur grille : routage par dimension ou éviter certains « tournants »
- Routage non déterministe : utiliser des canaux virtuels disposant de leurs propres ressources  
Ex. sur grille : attribuer les canaux par octant pour ne pas avoir de dépendance  
Difficile si dimension du réseau élevée
- Routages probabiliste : si conflict, dérouter un paquet. Attention aux *livelock*...



- Liaison à 10 Gb/s ( $4\times$ )
- Latence  $6 \mu\text{s}$
- Exemple de réseau de 512 nœuds <http://www.topspin.com> style *fat-tree*





- Elan4 QsNet II qui équipe Bull Tera-10 du CEA/DAM
- 912 Mo/s
- MPI sur Opteron
  - ▶ 1,5  $\mu$ s latence
  - ▶ <2  $\mu$ s barrière
- À la recherche du temps perdu...
  - ▶ 990 ns dans chipset processeur
  - ▶ 240 ns dans carte Elan
  - ▶ 218 ns dans câbles (vitesse de la lumière ☺)
  - ▶ 213 ns dans switches Elan
- <http://www.quadrics.com>
- $\approx \$1\ 700/\text{port}$  en 2006



- 10 Gb/s en Gigabit Ethernet ou Myrinet
- Myrinet : protocole plus efficace qu'Ethernet (entêtes...) :  
9,8 Gb/s, 2  $\mu$ s de latence MPI
- <http://www.myri.com>



- Le réseau de base pour les masses !
- La solution du pauvre...
- 1 Gb/s mais latence assez élevée (couches protocolaires)
- Utilisé sur machines haut de gamme comme lien d'administration



- Recherche débridée 1980-2000
- Dans la vraie vie actuelle : topologies simples à réaliser
  - ▶ Grilles 2D ou 3D
  - ▶ *Fat-tree* : réseaux multi-étage de routeurs favorisant localité
- Quelques constructeurs font encore du « sur mesure » (sur bus HyperTransport Opteron dans Cray XT3)



Si on n'accélère que les calculs : limite des E/S (AMDAHL...) !

Principalement :

- disques rapides (données ou mémoire virtuelle)
- bandes et disques optiques (données)
- sorties graphiques (plusieurs M polygones 3D/seconde)
- capteurs divers (acquisition de type CERN, physique des particules, avec 1 M capteurs!).



Augmenter débit et capacité mais diminuer coût  $\rightsquigarrow$  paralléliser les disques !

Problème : *Mean Time Between Failure* de plusieurs disques.

Endurance de  $N$  disques pendant un temps  $t$  :

$$R_N(t) = (R_1(t))^N$$

Est-ce bien utile ?

$$\lim_{N \rightarrow \infty} R_N(t) = 0$$

Si  $MTBF_1 = 30000$  heures, alors  $MTBF_{1000} = 30$  heures...

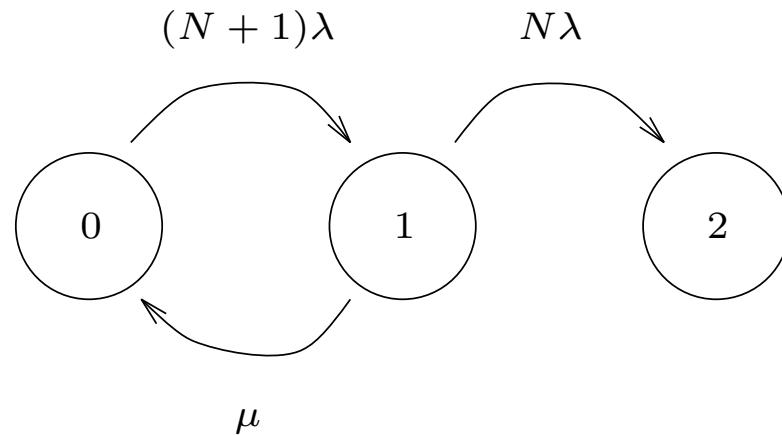


Mettre plus de disques pour compenser les pannes.

Chaîne de MARKOV modélisant un RAID où 0 ou 1 disque peut être en panne sans perte de données :

- $\lambda$  constante de panne d'un disque ( $1/MTBF_1$ )
- $\mu$  constante de réparation.

Supposition :  $\lambda \ll \mu$



Durée avant perte de données :

$$MTTDL \approx \frac{\mu}{N \times (N + 1)\lambda^2}$$



**RAID-0** : pas de redondance ! *stripping*

**RAID-1** : tout est doublé

- cher : moitié du disque utile
- rapide

**RAID-2** : rajouter  $C$  disques par  $D$  disques pour code de détection et correction d'erreur,  $C \geq \log_2(D + C + 1)$

**RAID-3,4,5** : si un contrôleur sait quand le disque est en panne (CRC sur disque, etc.)  $\rightsquigarrow$  seule la parité suffit.

Problème : tout accès nécessite un accès à la parité =: goulet d'étranglement !

**RAID-5** : répartition de la parité cycliquement sur les disques pour paralléliser les accès.



Conférence 2005 Design, Automation and Test in Europe

- Électronique partout. Automobiles
  - ▶ 30 % du prix d'une voiture haut de gamme est dans l'électronique
  - ▶ 90 % de l'innovation dans les voitures sera dans l'électronique (Daimler-Chrysler, 2000)
  - ▶ Plus d'électronique que dans tout le système Apollo
    - 75 processeurs
    - 150 moteurs électriques
    - 100 MLOC source
    - Besoin d'avoir une seule architecture pour tous les modèles du bas de gamme au haut de gamme
    - Évolutivité ↗ potentiellement des FPGA (Daimler-Chrysler)



- Besoin de performances mais d'économies d'énergie : mobile, systèmes communiquant sans fil et à pile,...
- Circuits intégrés immenses
- Besoin de rehiérarchiser les circuits
- Systems on Chip (SoC)
- Vieux problèmes se retrouvent au sein de chaque circuits
  - ▶ Multiprocessors on Chip (MPoC)
  - ▶ Network on Chip (NoC)
- Comment tester des systèmes avec 1+ Gtr, des programmes ?
- Notion d'IP (briques d'*Intellectual Property*)



*Centralized Run-Time Resource Management in a Network-on-Chip Containing Reconfigurable Hardware Tiles*, V. Nollet, T. Marescaux, P. Avasare, D. Verkest, J-Y. Mignolet. DATE2005.

- SoC avec des entités reconfigurables
- Placement de tâches « matérielles » à des endroits du circuit reconfigurable
- Allocation dynamique de nouvelles tâches
- Éventuellement besoin de bouger des tâches suite à la « fragmentation » de l'espace de fragmentation après des fins de tâches
- Définition de points de migration (équivalent au *check-pointing*)
- Migration de tâches sur le circuit alors que des paquets sont en route sur le NoC pour une tâche qui a bougée...
- Rajout de paquets de redirection dans le circuit



- Processeurs généralistes : optimisés pour opérations courantes
- Certaines applications ne fonctionnent pas forcément très bien sur ces processeurs prédéfinis
- ↗ Pour dépasser inefficacité : rajout de circuits logiques reconfigurables (programmables)
- Réalisent matériellement algorithmes voulus
- Très efficace en bioinformatique ou traitement d'image



- 144 Opterons
- Réseau spécifique sur canaux HyperTransport
- Synchronisation matérielle
- Cartes accélératrices à base de FPGA Xilinx Virtex 4.



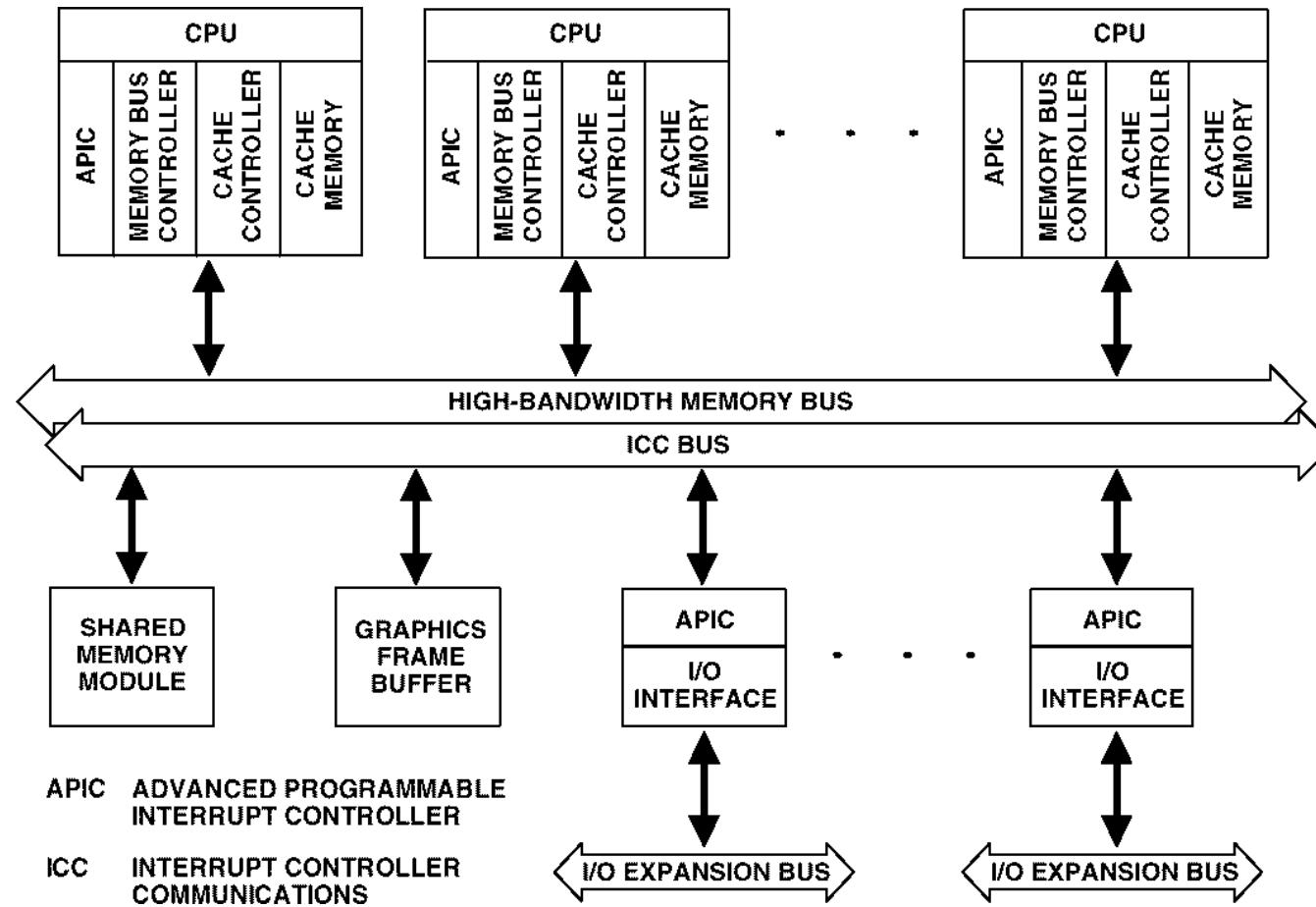
- Demande continue du grand public pour jeux vidéo toujours plus réalistes
  - ▶ Suréchantillonnage
  - ▶ Transluscence
  - ▶ Modèles d'illumination globale
  - ▶ ...
- ↗ Cartes d'accélérations graphiques extrêmement performantes
- Algorithmes en constante évolution ↗ Cartes graphiques ≡ véritables supercalculateurs
  - ▶ Beaucoup de mémoire
  - ▶ Spécialisées mais néanmoins programmables avec compilateurs C ou C++
  - ▶ Possible de faire travailler plusieurs cartes ensembles (technologie SLI de nVidia)

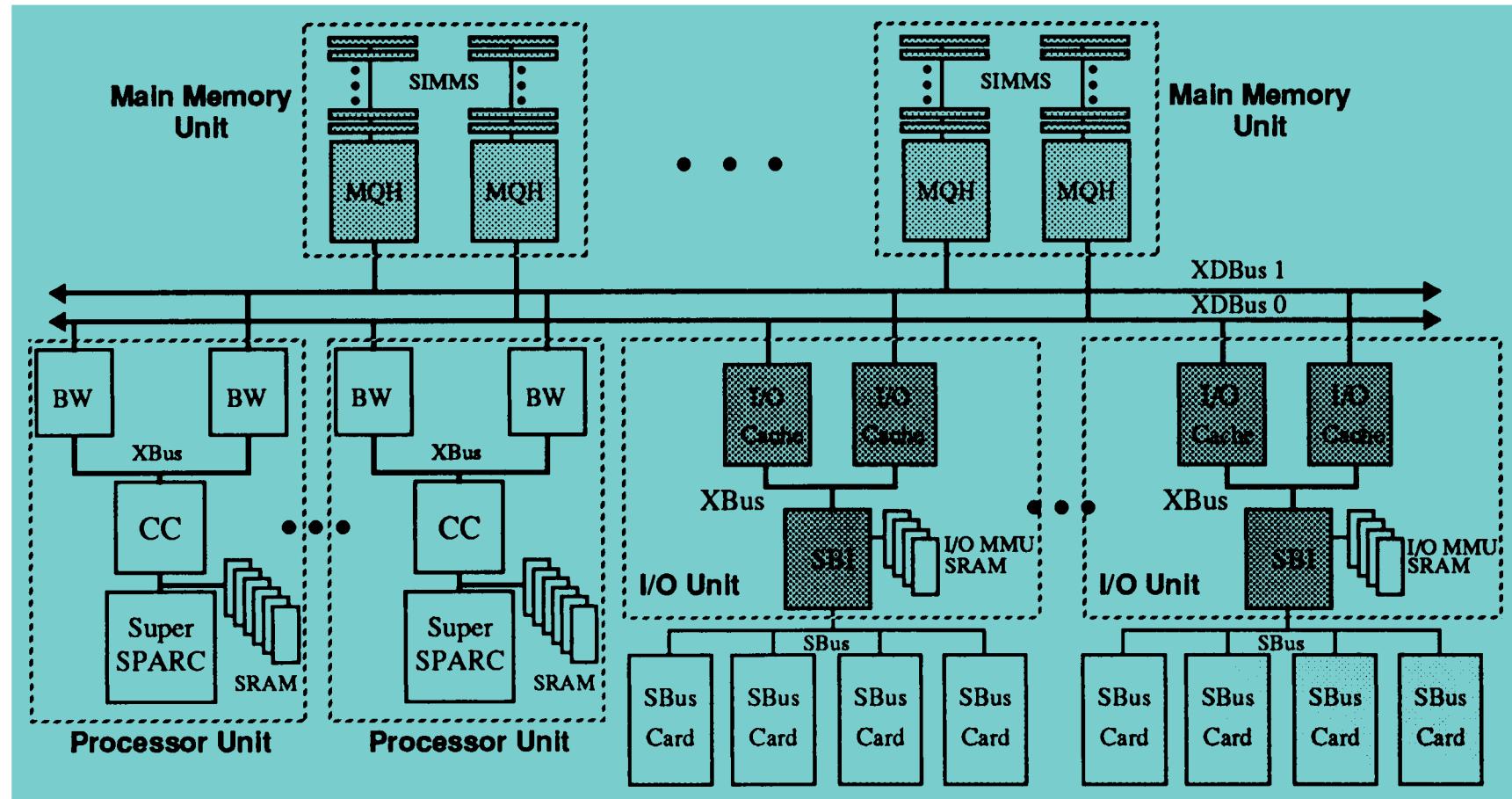


- Idée : utiliser pipelines de transformations géométriques pour calculs scientifiques



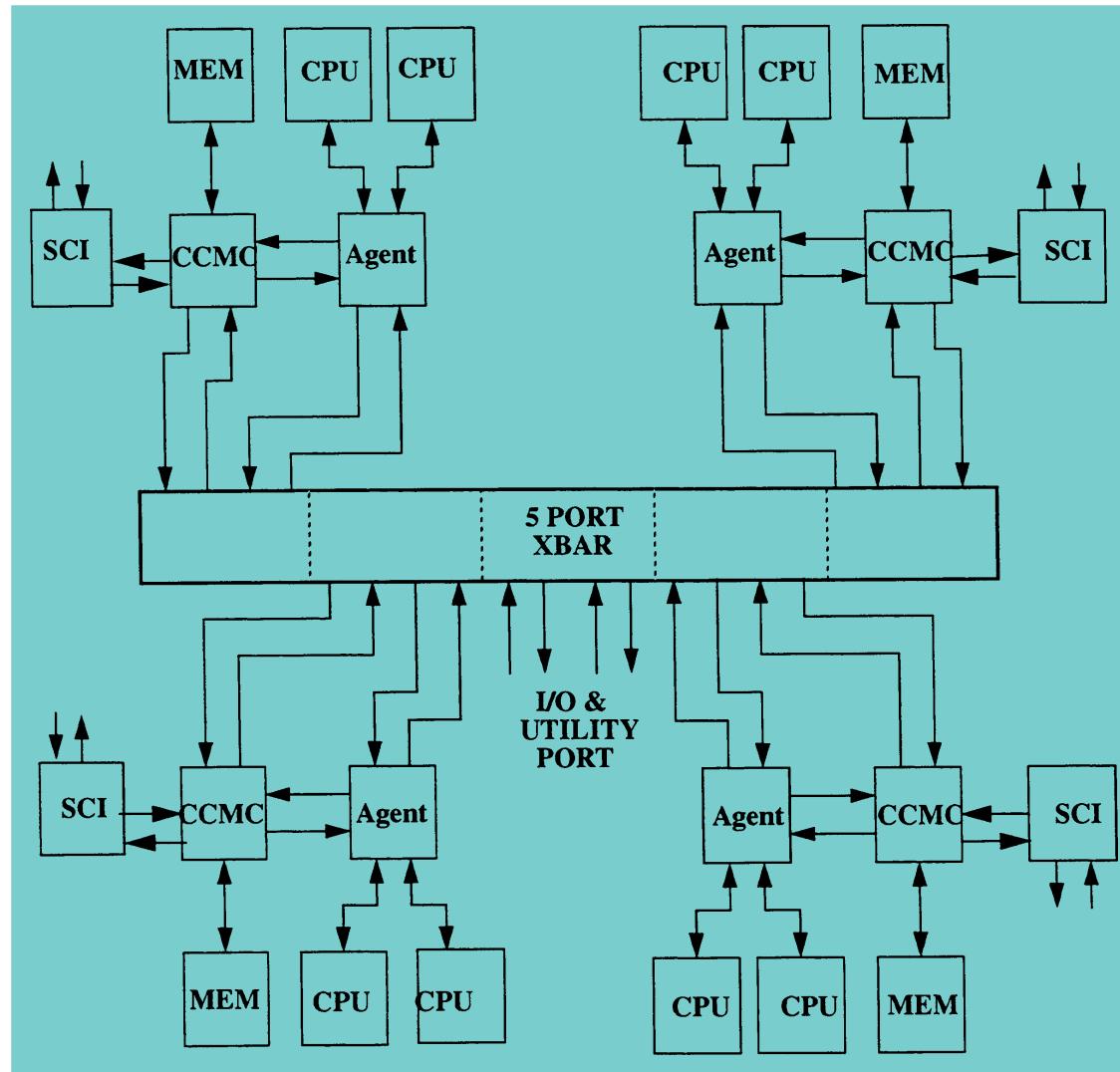
Pour plus de performances : plusieurs processeurs sur un même bus  
(MIMD à mémoire partagée)



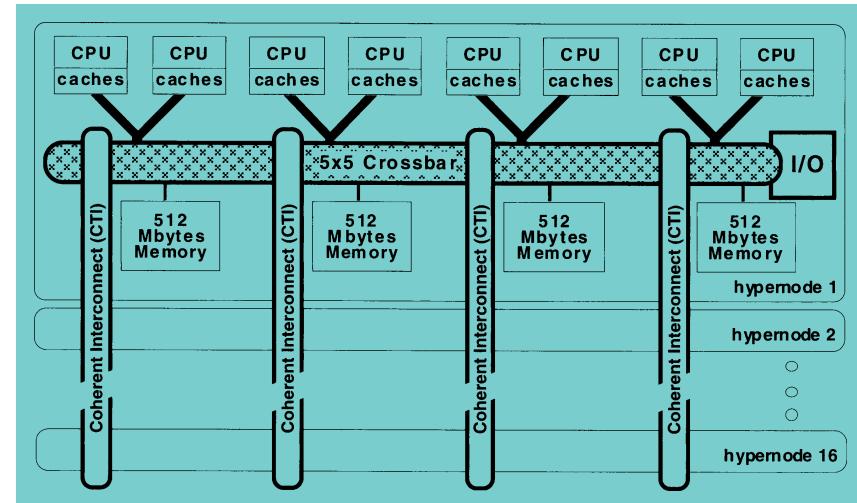


- 1,9 GFLOPS Linpack  $1000 \times 1000$
- SuperSPARC 60 MHz
- MIMD
- 2 XDBus
  - ▶ à commutation de paquets (latence mémoire)
  - ▶ 800 mV
  - ▶ 50 MHz & 64 bits/bus : 400 Mo/s/bus
  - ▶ pipelinable pour connexion de plusieurs XDBus
- projet Dragon de XEROX, suite de M0 du LIE à l'ENS
- Cray CS6400 : 64 SPARC sur 4 XDBus à 55 MHz

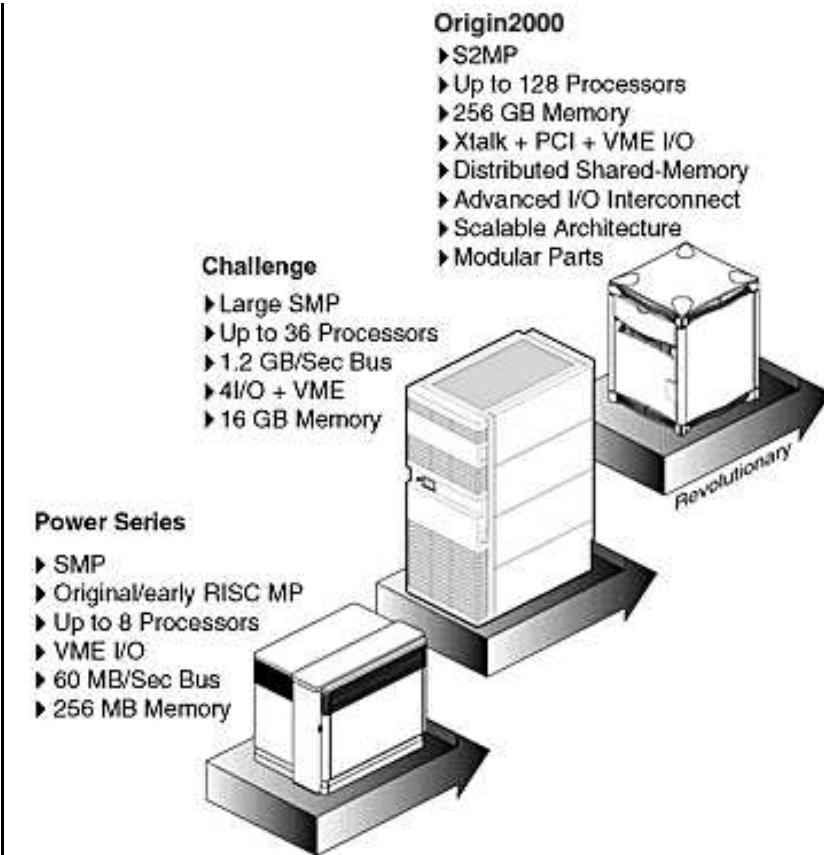


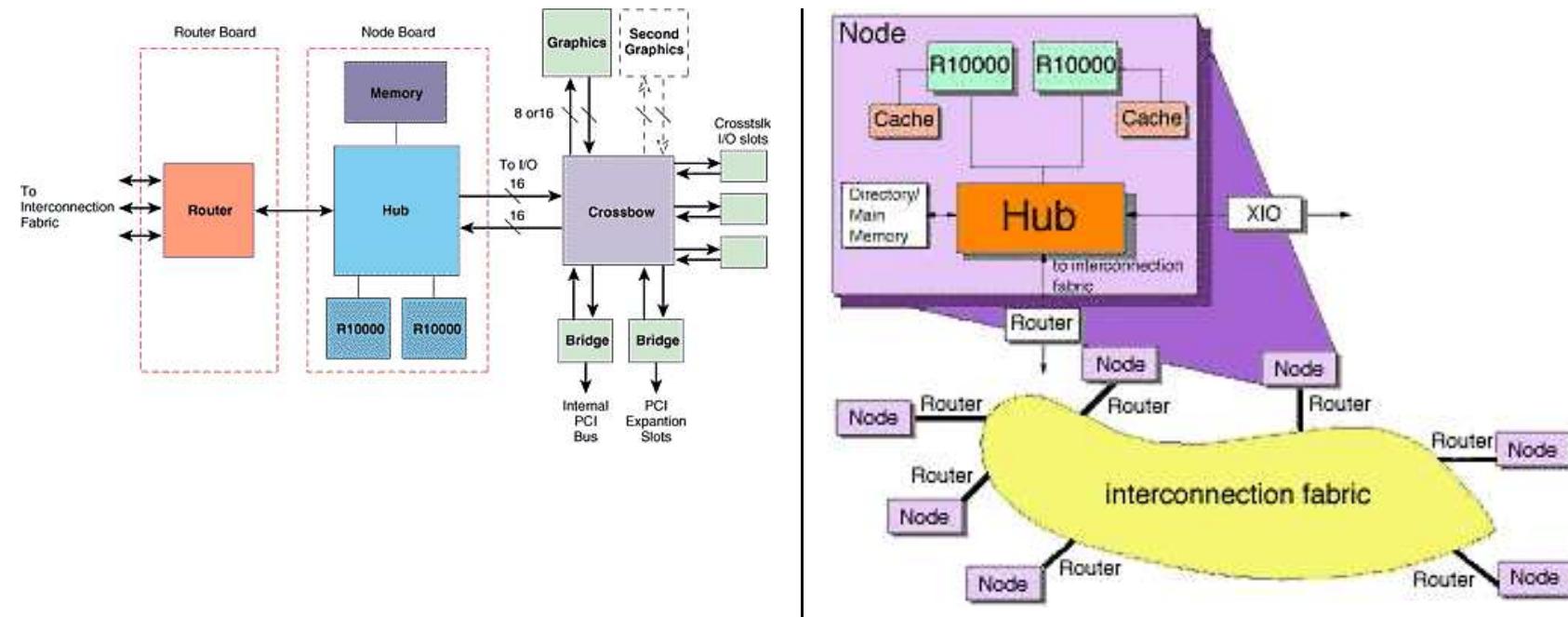


- hiérarchie : 16 hypernœuds de 4 modules de 2 HP PA7100 (1 → 128 PEs)
- MIMD
- 25,6 GFLOPS 64bits
- 256 Mo → 32 Go
- Réseau SCI (18 bits ECL différentiels,  $2 \times 250$  MHz)



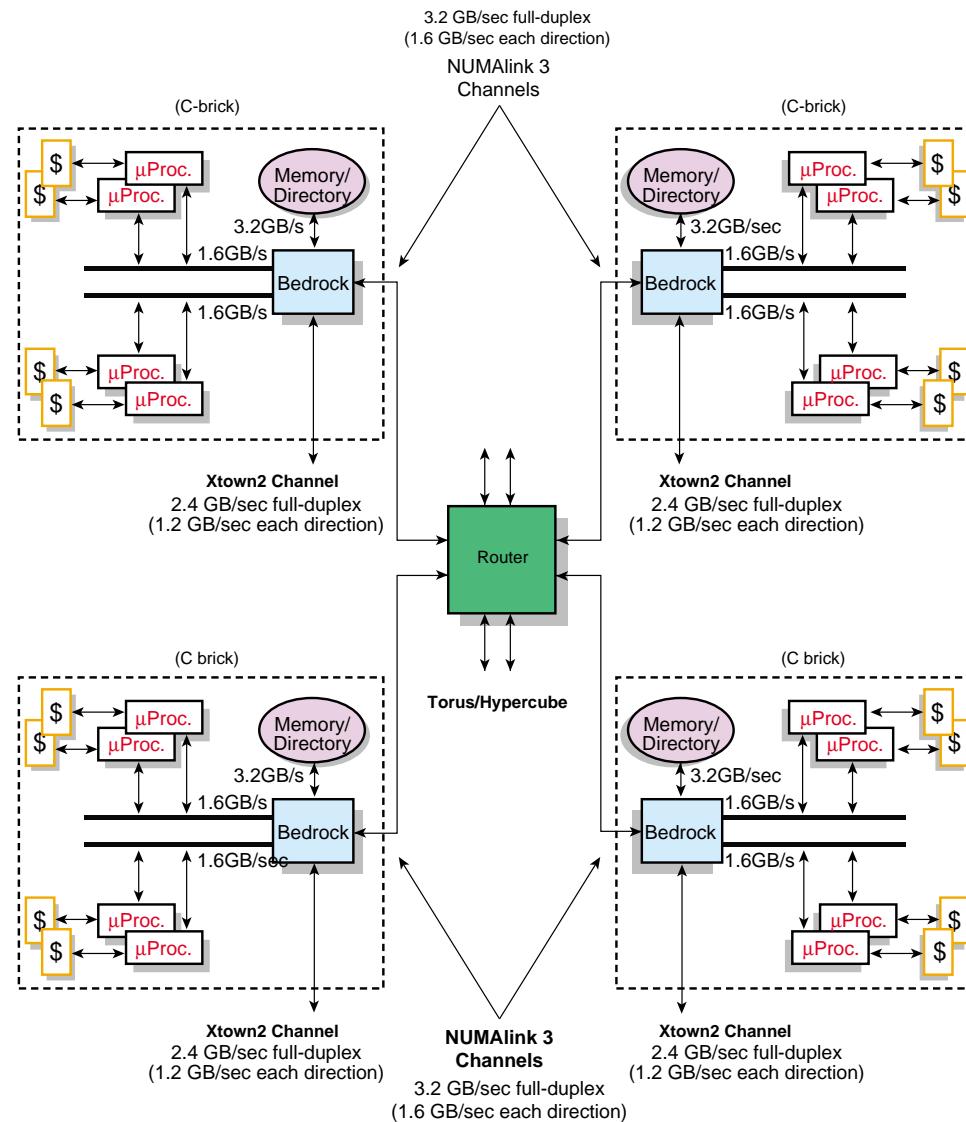
- Architecture hiérarchique : réseau de d'ordinateur à mémoire partagée
- Mémoire distribuée partagée
- 128 PEs R10000 200 MHz  
~~~ 51 GFLOPS

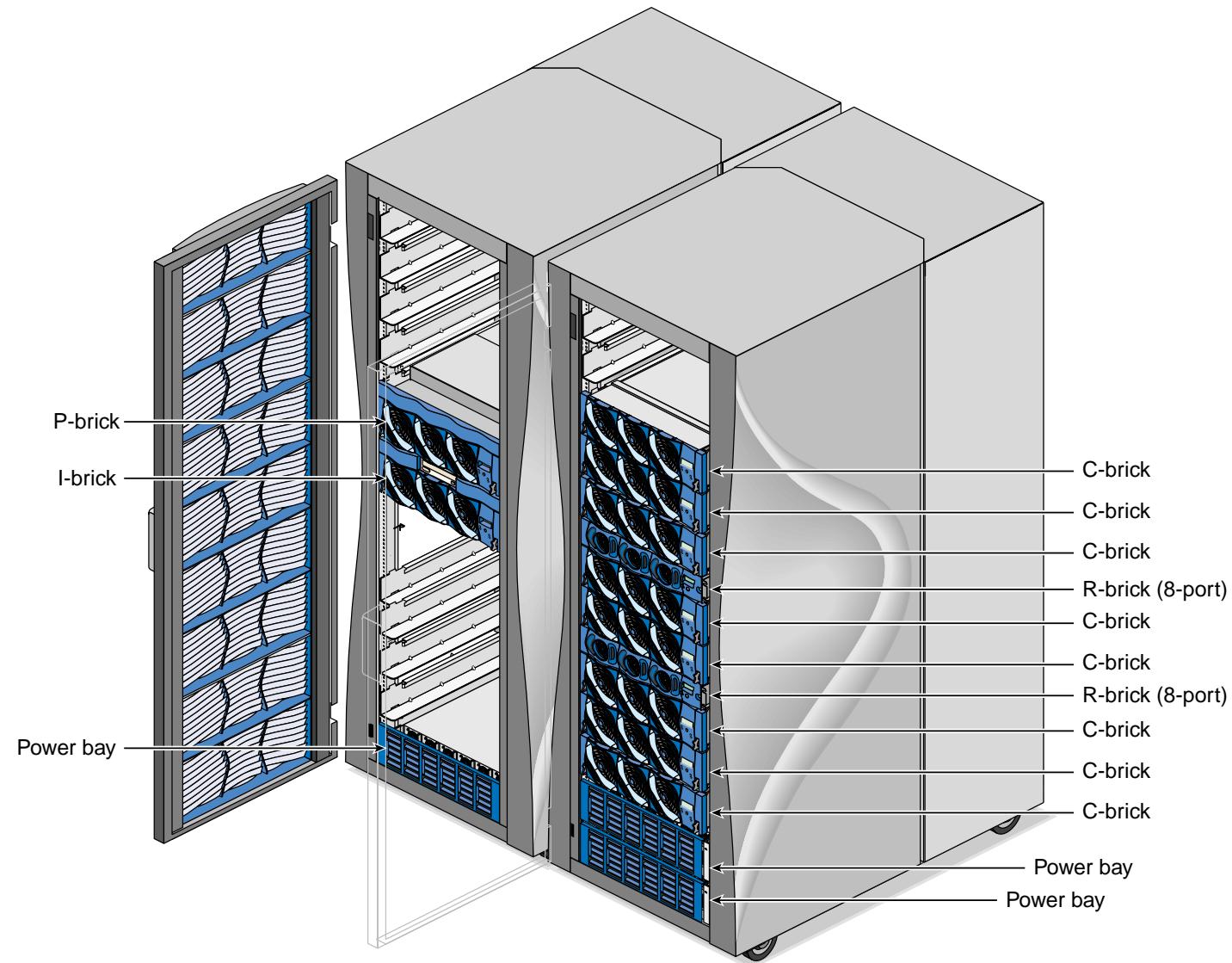




- Architecture hiérarchique : réseau de d'ordinateur à mémoire partagée
- Mémoire distribuée partagée (NUMAflex)
- Origin 3800
 - ▶ 512 PEs R14000A 600 MHz (499 SPEC CFP2000 Base),
2Go/PE
 - ▶ 1 To de mémoire
 - ▶ 16 armoires de 32 PEs
 - ▶ IRIX







| # CPUs | Router Hops
Max | Router Hops
Average | Local Memory
Latency | Worst-Case
Remote Latency | Average
Latency |
|--------|--------------------|------------------------|-------------------------|------------------------------|--------------------|
| 16 | 1 | 0.75 | 175 | 285 | 257.5 |
| 32 | 2 | 1.38 | 175 | 335 | 296.3 |
| 64 | 2 | 1.69 | 175 | 335 | 315.6 |
| 128 | 4 | 2.47 | 175 | 435 | 356.6 |
| 256 | 5 | 3.48 | 175 | 485 | 408.3 |
| 512 | 7 | 4.74 | 175 | 585 | 471.6 |

http://www.sgi.com/origin/3000/3000_ref.pdf



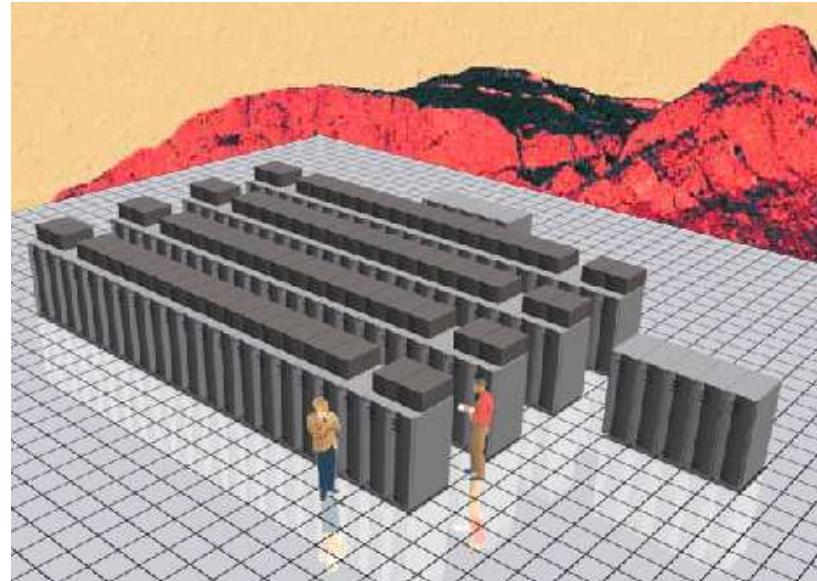
- Multifibre (*multithreaded*), 128 fibres
- PE 64 bits, 8 accès mémoires en cours/fibre
- 1 ou 2 Go/PE, 1 GFLOPS/PE
- Pas de cache de données !
- 8 Ko et 2 Mo de cache d'instruction
- Réseau tore 3D, 2,8Go/PE
- Mémoire partagée
- Peut supporter une latence mémoire de l'ordre de 500 cycles
- 256 PEs dans la grosse configuration
- 4kW évacués par eau/PE
- 10 ans de développements (une boîte vide à SuperComputing'95...)

<http://www.cray.com/products/systems/mta/>



<http://www.ssd.intel.com/sc95booth/tflop2.html>

- Utilisation de composants « de sur l'étagère »
- Système d'exploitation UNIX
- Vitesse $\times 10\,000$ par rapport à l'iPSC de 1985
- Vitesse $\times 1\,000\,000$ par rapport au CDC6600 de 1962 (1 MIPS)
- 160 m², 85 armoires, 800 kW
- 360 tonnes de climatisation
- Tripler les performances tous les 18 mois sur 10 ans



<http://www.intel.com/ebusiness/hpc>



IAHP

Département Informatique – ENST Bretagne

Architectures Parallèles

- ASCI Purple : 10 240 processeurs, N° 2 au Top 500 en 2005
- Version spéciale cluster (grappe)
- 8 Power5 1,9 GHz 64 bits ou 8 bi-cœur 1,5 GHz/lame
- 4 liaisons InfiniBand vers switches (TopSpin MPI...)
- 2 Ethernet 1 Gb/s
- AIX5L ou Linux



<http://www-03.ibm.com/servers/eserver/pseries/news/related/2004/m2040.pdf>

- 64 bits
 - Pipeline 15 étages
 - 8 instructions/cycle
 - SMT (*Simultaneous Multi-Threading*) à priorité pour remplir bulles du pipeline du Power4
 - 120 registres physiques entiers + 120 flottants partagés par les 32+32 registres virtuels des 2 threads : renommage à la volée style

```
r3 = r1 + r2      =: r3 = r1 + r2; r1' = r4 * r5
r1 = r4 * r5
```

- Centaines de compteurs de performance pour comprendre ce qui se passe 😊





- 60 TFLOPS LINPACK au CEA/DAM
- 544 Bull Novascale 6160 avec 8 Intel Montecito double cœur
- 27 To de mémoire
- 54 serveurs d'E/S
- 1 Po de disques
- Linux & système de fichiers Lustre
- Réseau Quadrics

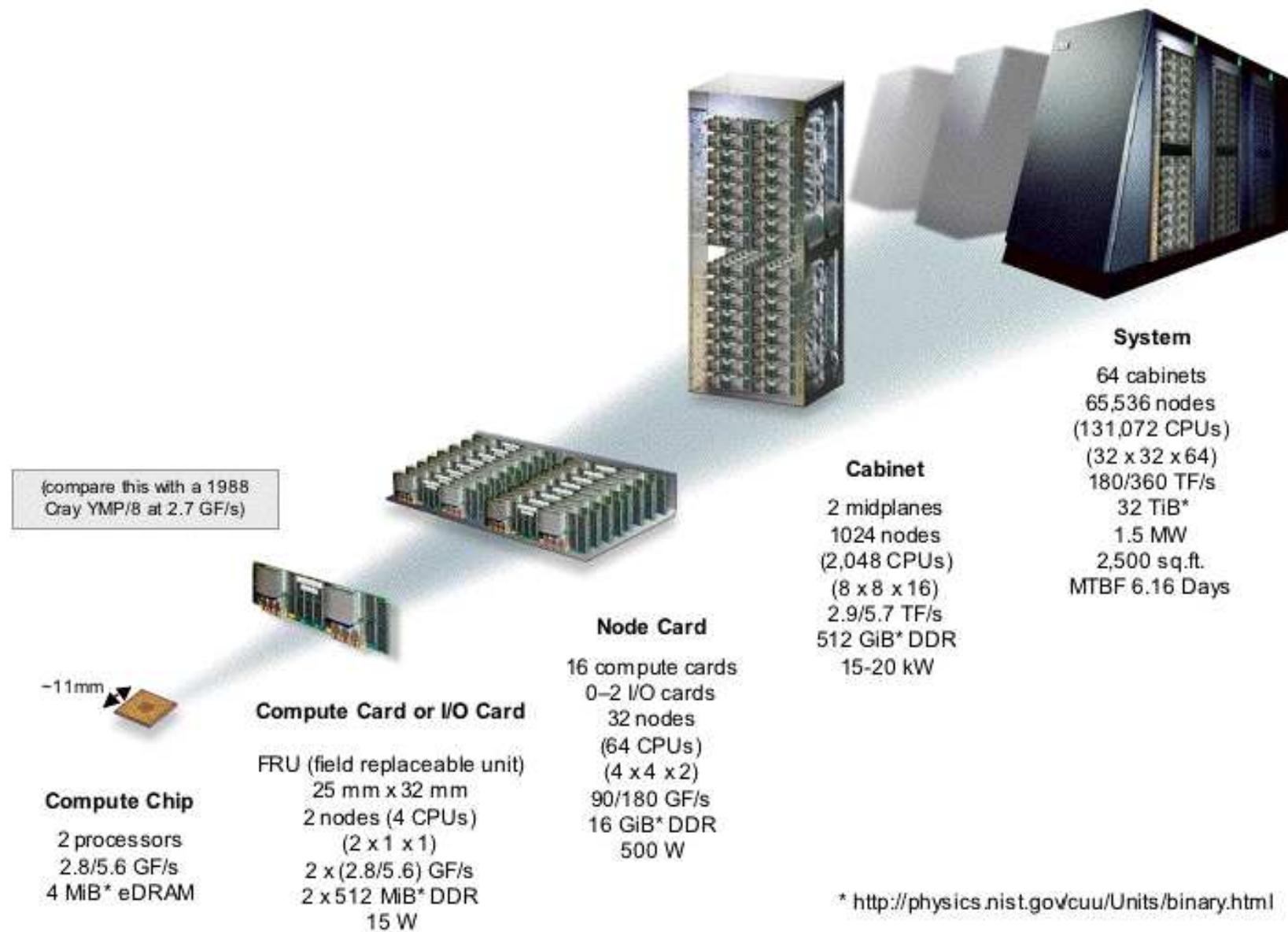


<http://www.llnl.gov/asc/platforms/bluegenel/overview.html>

<http://www.llnl.gov/asc/platforms/bluegenel/arch.html>

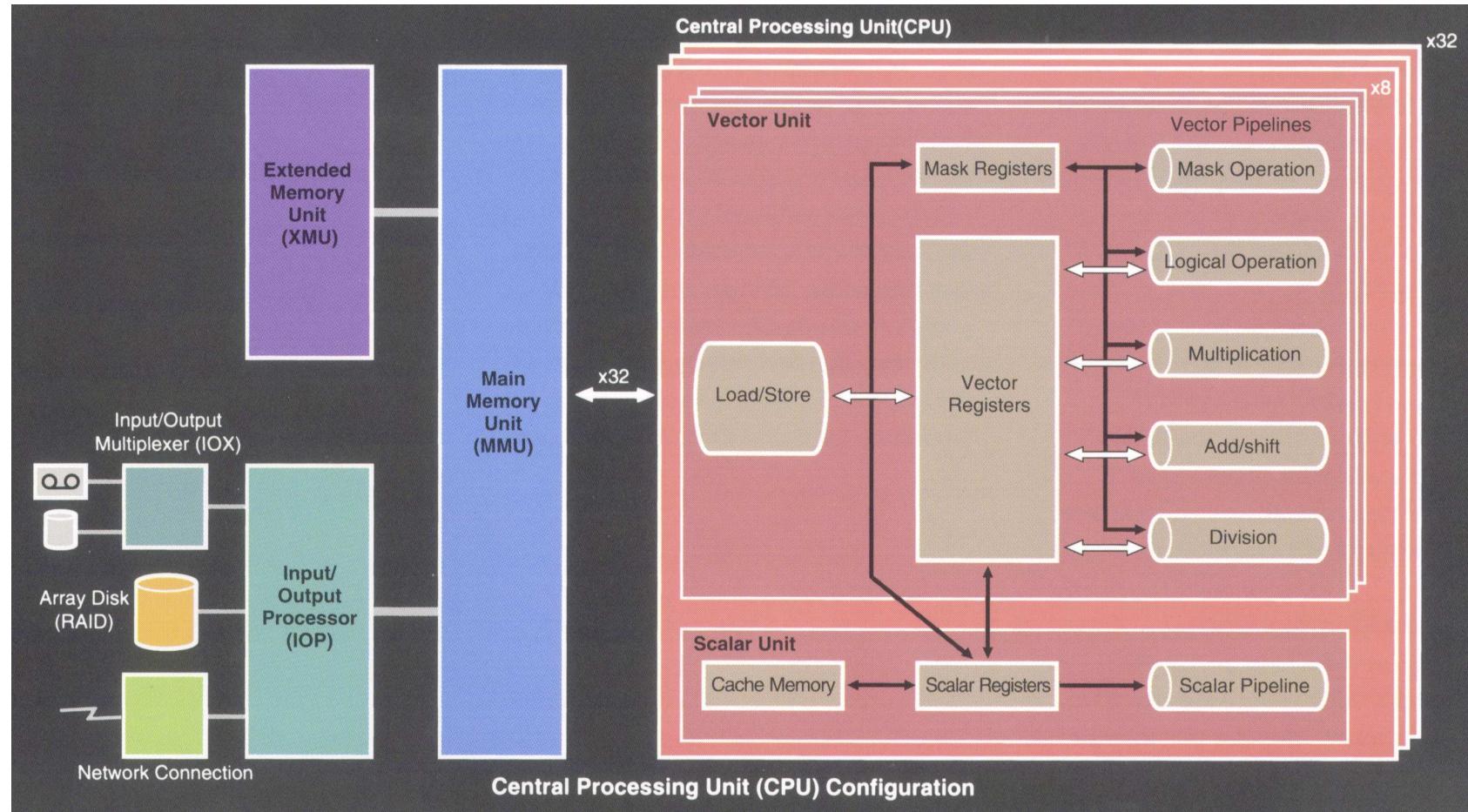
- N° 1 au Top 500 en 2005 avec 131 072 processeurs
- Base de 2 PowerPC440 700 MHz avec 2 unités de calcul flottant
- 10× efficacité électrique par rapport aux pSeries
- Réseau tore 3D + arbre pour réductions/diffusions
 - ▶ Diamètre 64
 - ▶ Latence inter-nœud de 100 ns
 - ▶ 6,4 ns latence maximum
 - ▶ 175 Mo/s/lien assez faible

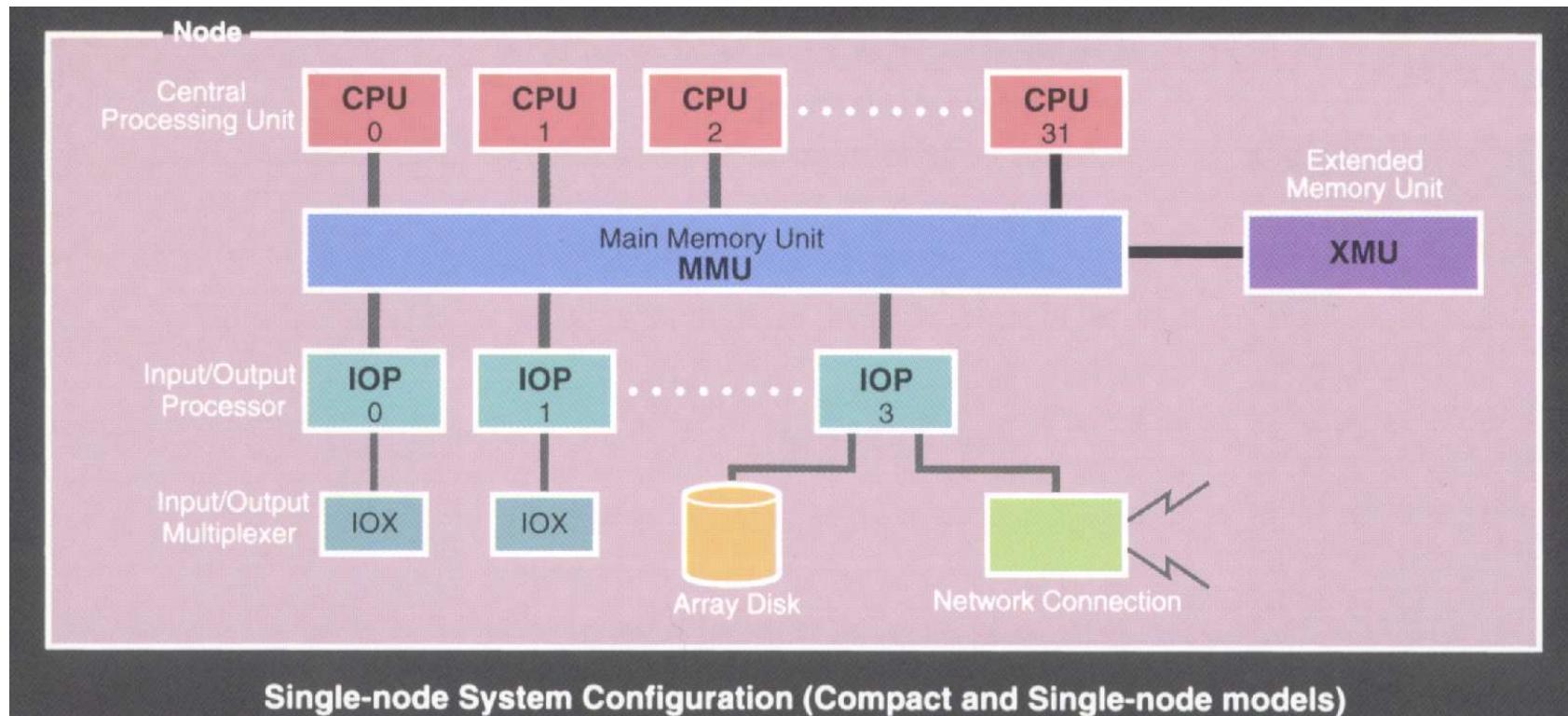


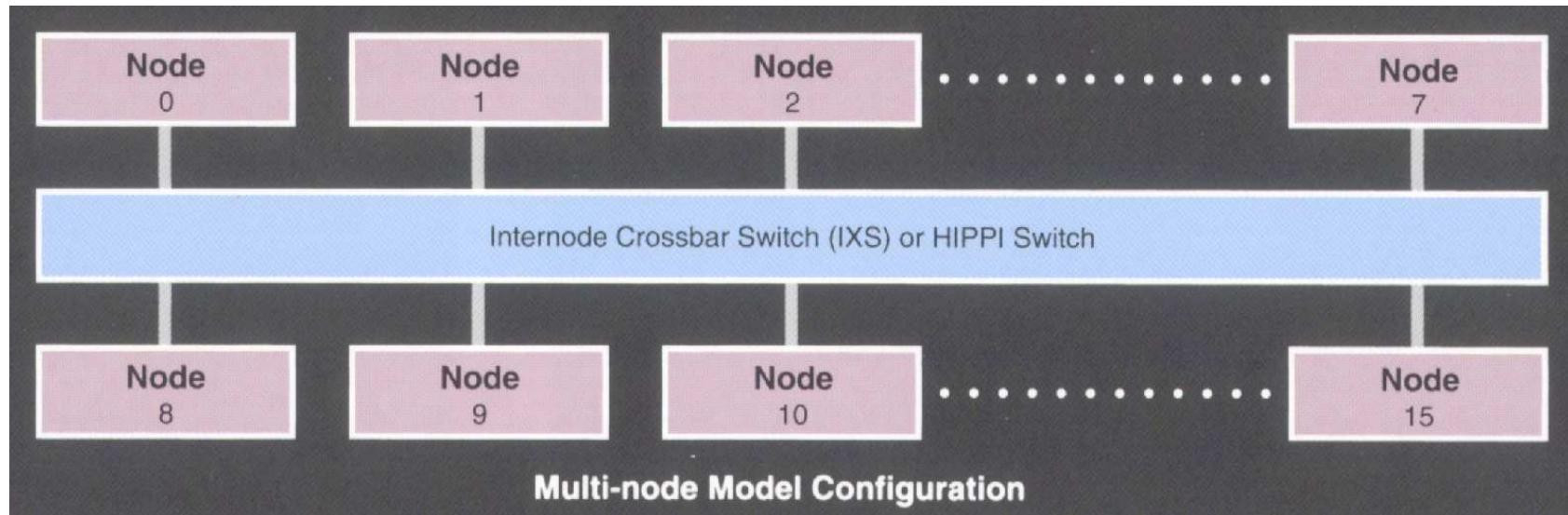


- MTBF de BlueGene/L : 6 jours...
- 1951 : simulateur temps réel Whirlwind de Jay FORRESTER & Bob EVERETT
 - ▶ 500 000 +/s, 50 000 ×/s
 $5,6 \cdot 10^{-9} \times$ BlueGene/L ☺ ↗ +51 %/an en 54 ans
 - ▶ Mémoire à tores
 - ▶ Lampes ↗ consomme \$32 000 de tubes/mois ! ☺
- ↗ Faire du check-pointing & redondance
- ↗ Projet ANR ARA SSIA SafeScale plus général prenant en compte attaques malicieuses dans grilles (ENSTB-IMAG-Paris 13-IRISA)









- 2,5 ns de temps de cycle, CMOS
- 2 GFLOPS/PE & 8 pipelines vectoriels/PE
- 16 Go/s/PE
- 1 GFLOPS → 1 TFLOPS
- Registres vectoriels partitionnables
- 2-128 Go max de mémoire *statique synchrone*



- Pas de cache de données
- Mémoire partagée par crossbar
- hiérarchie 512 PEs (16 nœuds autour d'un crossbar) dans la grosse configuration
- MIMD vectoriel
- 76,8 Go/s d'E/S



- 8 GFLOPS/PE & 8 pipelines vectoriels/PE
- 1 nœud = 8 PEs
- 256 Go/s/nœud et 64 Go/nœud
- 8 To max de mémoire SDRAM DDR
- hiérarchie 1024 PEs (128 nœuds autour d'un crossbar)
- Mémoire partagée par crossbar
- 8 TFLOPS
- MIMD vectoriel à mémoire partagée
- 1 To/s d'E/S
- Refroidissement à air

<http://www.sw.nec.co.jp/hpc/sx-e/sx6>



- Processeurs vectoriels de 16 GFLOPS : 1 processeur scalaire + 4 processeurs vectoriels
- 8 processeurs par nœud
- 512 nœuds = 65 TFLOPS
- Processeur CMOS mono-chip 90 nm & 9 niveaux de cuivre
- 8 210 pattes dont 1 923 de signaux !
- Record du monde de 300 Go/s circuit-extérieur



- Cray Research créé en 1972 par Seymour Cray
- Premier Cray-1 installé à Los Alamos National Laboratory en 1976 : 8.8 M\$, 160 MFLOPS, 8 Mo, ECL
- 1982 : Cray X-MP
- 1985 : Cray-2
- 1988 : Cray Y-MP avec 1 GFLOPS efficace (2,3 GFLOPS crête)
- 1989 : conflit technologique ↗ Cray fonde Cray Computer Corporation pour développer le Cray-3 en AsGa. Projet jamais débogué ↗ démarrage du Cray 4
- Cray Research introduit série C90 (1 GFLOPS/PE) et Y-MP EL (version CMOS)
- 1993 : ordinateur massivement parallèle T3D à base d'Alpha
- 1995 : T3E



- 1996 : décès de Seymour Cray (71 ans) suite à un accident de voiture
- 1996 : fusion de Cray Research avec SGI
- 1998 : T3E-1200 premier à atteindre 1 TFLOPS sur une application
- 2000 : Cray Research vendu à Tera Computer Company



- 100 % AsGa DCFL
- 1 ns de temps de cycle
- 2 GFLOPS/PE
- 16 Go/s/PE
- Modules reliés par micro-coaxiaux
- 64 Go max de mémoire *statique*, 4 Mbits, 21 ns
- Pas de cache de données !
- Tampon d'instruction à la CDC6600
- Mémoire partagée par crossbar
- 128 PEs dans la grosse configuration
- 4kW évacués par eau/PE

Une boîte vide à SuperComputing'94... Abandonné car trop difficile à mettre au point



Ratisse large !

- Cray MTA (1996) : machine Tera
- Cray SV1 (1999) : vectoriel + MPP
- Cray SX-6 (2002) : revend le NEC SX-6
- Cray T3E (1996) : MPP
- Cray X1 (2003) : vectoriel + MPP
- Cray HPC Cluster : à mémoire distribuée sous Linux



- Multi-Streaming Processor (CMOS 500 MHz) : visible en 4 PEs de 2 GFLOPS ou processeur vectoriel 8 GFLOPS à 8 tuyaux
- SDRAM
- UNICOS
- Architecture hétérogène
- SV1 ex-4
 - ▶ 4 nœuds SMP
 - ▶ 24 MSP
 - ▶ 32 Single-Streaming Processors de 2 GFLOPS
 - ▶ 96 Single-Streaming Processors « normaux »
 - ▶ 256 GFLOPS
 - ▶ 128 Go
 - ▶ + 128–384 Go de mémoire secondaire SSD



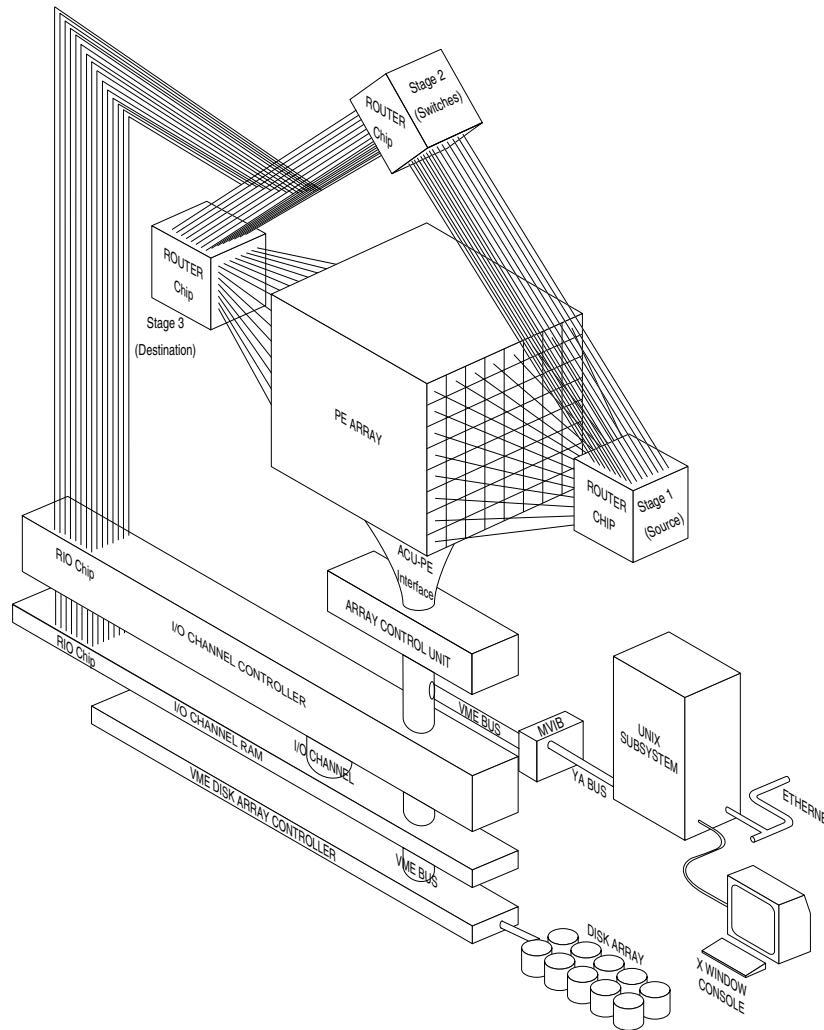
- Processeur Alpha 21164A (EV5.6), 4-superscalaire, 675 MHz, 2 opérations flottantes par cycle
- Refroidissement à eau
- NUMA et interconnexion tore 3D
- DRAM 50 ns
- T3E-1350 LC2176
 - ▶ 20,8 m² ☺
 - ▶ 8 baies
 - ▶ 2176 PEs par paquets de 8
 - ▶ 3 TFLOPS
 - ▶ 544 Mo—1 To



- Processeurs vectoriels de 18 GFLOPS : 4 processeurs vectoriels
- 16 à 8 192 ↵ 147 TFLOPS
- Mémoire partagée
- Réseau avec 16 tores 2D

<http://www.cray.com/products/x1e>





- 16K PEs 32 bits
- SIMD & programmation data-parallèle
- 12,5 MHz
- 4,2 MIPS/PE (12,5 crête)
- Mémoire distribuée
- 256 Ko/PE
- 1 bus mémoire/16 PEs ↗ 23 Go/s (1,5 Mo/s/PE)
- Accès mémoires indirect ralentis d'un facteur 3



- Prendre des PC rapides sans écran
- Acheter des cartes d'interconnexions rapides
(<http://www.lri.fr/~fci/RS.html>, Myrinet, 1 Gbit/s)
- Brancher & allumer
- Récupérer un compilateur parallèle du domaine public (suivre <http://www.cri.ensmp.fr/pips>)
- Installer bibliothèque de communication
- Programmer...
- Utiliser...
- Recycler !



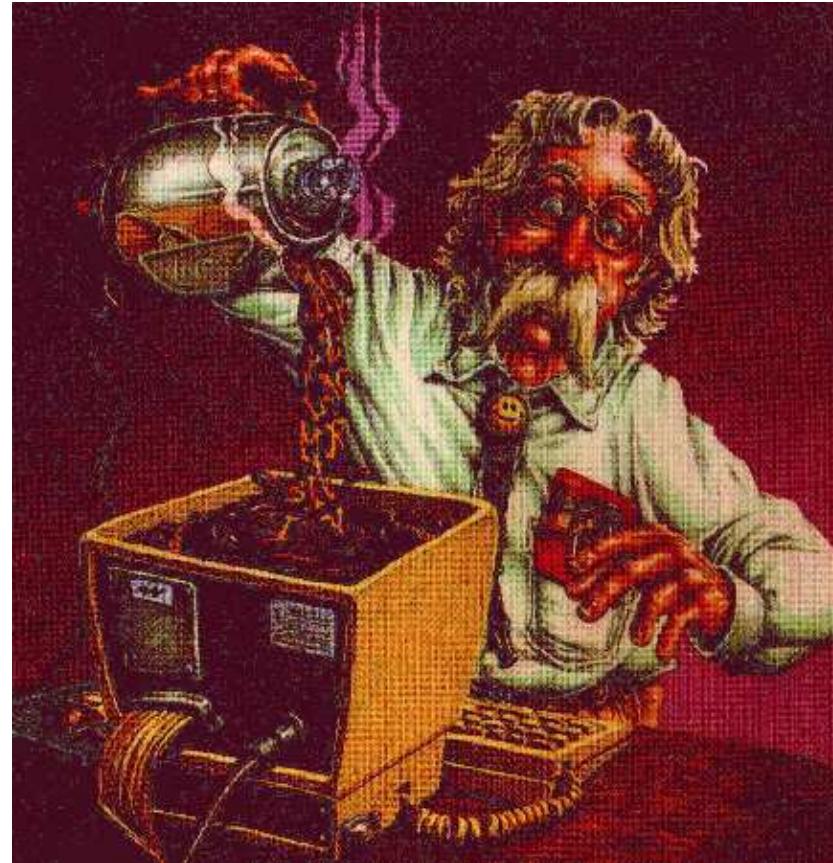
- De plus en plus d'applications utilisent les supercalculateurs
- Unix règne en maître absolu
- Toujours plus de puissance
- Vitesse de calcul aujourd'hui : rassemblement de toutes les techniques du passé
 - ▶ Travail à la chaîne (pipeline)
 - ▶ Hiérarchie mémoire (cache)
 - ▶ Parallélisme intra- et interprocesseur
 - ▶ Nombre de processeurs ↗
 - ▶ Superscalaires voire vectoriels, instructions SIMD, multi-cœurs (même sur portables) ↘ démocratisation
 - ▶ Grand vainqueur de l'informatique : microprocesseur standard (RISC & superscalaire) et utilisation des technologies



- ~~> Convergence vers une taille et un nombre de PEs vers les hautes performances
- ▶ Coprocesseurs graphiques, reconfigurables
- ▶ Réseaux : simples (grilles), voire sur étagère (Ethernet)
- ▶ Retour des machines virtuelles des années 70 : virtualisation des machines parallèles avec différents OS...
- ~~> Architectures de plus en plus hétérogènes
 - ▶ Outils automatiques peu efficaces généralement dans vraie vie
 - ▶ Complexité pour le programmeur
 - ▶ Diversité architecturale ~~> nivellement par le bas du modèle de programmation : passage de message (MPI ≡ assembleur du parallélisme)



- Vers des ordinateurs pétaflopiques et exaflopiques...
- Comment rester proche puissances crêtes annoncées ?



Retour à compromis de modèles de programmation hétérogènes pour architecture hétérogène

- Machines moyennement parallèles : mémoire partagée intéressante car + simple à programmer
- Nœuds SMP programmés en OpenMP en interne (multi-thread pour les nuls ☺)
- Interconnexion de ces nœuds programmés en MPI (passage de messages pour les nuls ☺)
- Code efficace sur des machines parallèles modernes : connaître *encore plus* l'architecture !
- Beaucoup de travail de compilation
- Parallélisme **inter**processeur ? En dernier recours !



- Pour programmeurs
 - ▶ Maîtriser complexité globale + complexité applications
 - ▶ Tolérer latence mémoire (NUMA) + réseaux (GRID)
- Architectes
 - ▶ Machines efficaces simplement
- Spécialistes en compilation
 - ▶ Créer chaînon manquant !
 - ▶ Fournir outils plus efficaces et de plus haut niveau



Références

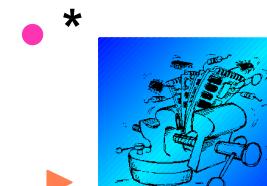
- [Akers et al., 1987] Akers, S. B., Harel, D., and Krishnamurhy, B. (1987). « The Star Graph : An Attractive Alternative to the n -Cube ». In *International Conference on Parallel Processing*, pages 393–400. The Institute of Electrical and Electronics Engineers, Inc., Academic Press.
- [Akl et al., 1992] Akl, S. G., Cosnard, M., and Ferreira, A. G. (1992). « Data-Movement-Intensive Problems : Two Folk Theorems in Parallel Computation Revisited ». *Theoretical Computer Science*, 95(2).
- [Allen et al., 1983] Allen, J. R., Kennedy, K., Porterfield, C., and Warren, J. (1983). « Conversion of Control Dependence to Data Dependence ». In *Conference Record of the Tenth Annual ACM Symposium on Principles Of Programming Languages*, pages 177–189. Association for Computing Machinery.
- [Alliant, 1991] Alliant (1991). « *The Campus/800 System from Alliant* ». Alliant Computer System Corporation. BR05-7915M.
- [Amdahl, 1967] Amdahl, G. M. (1967). « Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities ». In AFIPS, editor, *Proceedings of the Spring Joint Computer Conference*, pages 483–485.
- [Balakrishnan et al., 1988] Balakrishnan, M., Jain, R., and Raghavendra, C. S. (1988). « On Array Storage For Conflict-Free Memory Access For Parallel Processors ». In *Proceedings of the 1988 International Conference on Parallel Processing*, pages 103–107. The Pennsylvania State University Press.
- [Barnes et al., 1968] Barnes, G. H., Brown, R. M., Kato, M., Kuck, D. J., Slotnick, D. I., and Stokes, R. A. (1968). « The ILLIAC IV Computer ». *IEEE Transactions on Computers*, C-17(8) :746–757.
- [Batcher, 1968] Batcher, K. E. (1968). « Sorting networks and their applications ». In *The Proceedings of AFIPS 1968 SJCC*, pages 307–314. AFIPS Press.
- [Beneš, 1962] Beneš, V. E. (1962). « On a Class of Multistage Interconnection Networks ». *The Bell System Technical Journal*, XLI(5) :1481–1492.
- [Bermond and Fourneau, 1988] Bermond, J.-C. and Fourneau, J.-M. (1988). « Independent connections : an easy characterization of baseline-equivalent multistage interconnection networks ». In *International Conference on Parallel Processing*, pages 187–190. The Institute of Electrical and Electronics Engineers, Inc., Academic Press.
- [Blank, 1990] Blank, T. (1990). « The MasPar MP-1 Architecture ». In IEEE, editor, *IEEE Compcon Spring 1990*.
- [Bull, 1957] Bull (1957). « *Gamma 60* ». Compagnie des Machines Bull. 105.150.
- [Bull, 1992] Bull (1992). « *Naissance d'un Ordinateur (Bull DPS 7000) à livre ouvert* ». Réseaux et Systèmes d'Information — Bull.
- [Campbell et al., 1992] Campbell, L., Carlsson, G. E., Dinneen, M. J., Faber, V., Fellows, M. R., Langston, M. A., Moore, J. W., Mullhaupt, A. P., and Sexton, H. B. (1992). « Small Diameter Symmetric Networks from Linear Groups ». *IEEE Transactions on Computers*, 40(2) :218–220.
- [Chen, 1983] Chen, S. (1983). « Large-scale and High-Speed Multiprocessor System for Scientific Applications — CRAY X-MP-2 Series ». In *Proceedings of NATO Advanced Research*



- [Workshop on High Speed Computing, pages 46–58. IEEE Computer Society Press. Voir [Hwang, 1984].]
- [Clos, 1953] Clos, C. (1953). « A Study of Non-Blocking Switching Networks ». *The Bell System Technical Journal*, XXXII :406–424.
- [Cray, 1991] Cray (1991). « *The CRAY Y-MP C90 Supercomputer System* ». Cray Research, Inc. MCPB-104-1191.
- [Ein-Dor, 1985] Ein-Dor, P. (1985). « Grosch's Law Re-Revisited : CPU Power and the Cost of Computation ». *Communications of the ACM*, 28(2) :142–151.
- [Faber et al., 1986] Faber, V., Lubeck, O. M., and White, Jr., A. B. (1986). « Superlinear Speedup of an Efficient Sequential Algorithm is not Possible ». *Parallel Computing*, 3(3) :259–260.
- [Faber et al., 1987] Faber, V., Lubeck, O. M., and White, Jr., A. B. (1987). « Comments on the paper “Parallel Efficiency can be Greater than Unity” ». *Parallel Computing*, 4 :209–210.
- [Feng, 1974] Feng, T.-Y. (1974). « Data Manipulating Functions in Parallel Processors and Their Implementations ». *IEEE Transactions on Computers*, C-23(3) :309–318.
- [Fisher, 1981] Fisher, J. A. (1981). « Trace Scheduling : A Technique for Global Microcode Compaction ». *IEEE Transactions on Computers*, C-30(7) :478–490.
- [Flynn, 1966] Flynn, M. J. (1966). « Very High-Speed Computing Systems ». *Proceedings of the IEEE*, 54(12) :1901–1909.
- [Flynn, 1972] Flynn, M. J. (1972). « Some Computer Organizations and Their Effectiveness ». *IEEE Transactions on Computers*, C-21(9) :948–960.
- [Foster and Riseman, 1972] Foster, C. C. and Riseman, E. M. (1972). « Percolation of Code to Enhance Parallel Dispatching an Execution ». *IEEE Transactions on Computers*, C-21(12) :1411–1415.
- [Frailong et al., 1985] Frailong, J., Jalby, W., and Lefant, J. (1985). « XOR-Schemes : a Flexible Data Organization in Parallel Memories ». In INRIA, editor, *Cours et séminaires, conférences, Saint-Cyprien*, pages 33–45.
- [Gajski et al., 1987] Gajski, D., Milutinović, V. M., Siegel, H. J., and Furht, B. (1987). *Tutorial : Computer Architecture*. IEEE Computer Society Press.
- [Gustafson, 1988] Gustafson, J. L. (1988). « Reevaluating Amdahl's Law ». *Communications of the ACM*, 31(5) :532–533.
- [Hennessy et al., 1990] Hennessy, J. L., Patterson, D. A., and Goldberg, D. (1990). *Computer Architecture — A Quantitative Approach*. Morgan Kaufmann Publishers, Inc.
- [Hwang, 1984] Hwang, K. (1984). *Tutorial : Supercomputer : Design and Applications*. IEEE Computer Society Press.
- [INMOS, 1991] INMOS (1991). « *The T9000 Transputer Products Overview Manual* ». inmos — SGS-TOMSON, first edition.
- [Intel, 1991] Intel (1991). « *A Touchstone DELTA System Description* ». Intel Corporation. Advanced Information.
- [Kennedy and McKinley, 1990] Kennedy, K. and McKinley, K. S. (1990). « Loop Distribution with Arbitrary Control Flow ». In *Proceedings of Supercomputing '90*, pages 407–416. The Institute of Electrical and Electronics Engineers, Inc.



- [Keryell and Paris, 1993] Keryell, R. and Paris, N. (1993). « Activity Counter : New Optimization for the Dynamic Scheduling of SIMD Control Flow ». In *1993 International Conference on Parallel Processing — Volume II : Software*, pages II-184–II-187, Saint Charles, Ohio, USA.
- [Kim et al., 1991] Kim, S.-D., Nichols, M. A., and Siegel, H. J. (1991). « Modeling Overlapped Operation between the Control Unit and Processing Elements in an SIMD Machine ». *Journal of Parallel and Distributed Computing*, 12(4) :329–342.
- [Koppelman and Oruç, 1990] Koppelman, D. M. and Oruç, A. Y. (1990). « A Self-Routing Permutation Network ». *Journal of Parallel and Distributed Computing*, 10(2) :140–151.
- [Kuck, 1976] Kuck, D. J. (1976). « *Advances in Computers* », volume 15, Chapitre Parallel Processing of Ordinary Programs, pages 119–179. Academic Press.
- [Lawrie, 1975] Lawrie, D. H. (1975). « Access and Alignment of Data in an Array Processor ». *IEEE Transactions on Computers*, C-24(12) :1145–1155.
- [Lawrie and Vora, 1982] Lawrie, D. H. and Vora, C. R. (1982). « The Prime Memory System for Array Access ». *IEEE Transactions on Computers*, C-31(5) :435–442.
- [NEC, 1991] NEC (1991). « SX-3R series ». NEC Supercomputer. Cat. N° E51473 92022001KP.
- [Preparata and Vuillemin, 1981] Preparata, F. P. and Vuillemin, J. (1981). « The Cube-Connected Cycles : A Versatile Network for Parallel Computation ». *Communications of the ACM*, 24(5) :300–309.
- [Riseman and Foster, 1972] Riseman, E. M. and Foster, C. C. (1972). « The Inhibition of Potential Parallelism by Conditional Jumps ». *IEEE Transactions on Computers*, C-21(12) :1405–1411.
- [Russel, 1978] Russel, R. M. (1978). « The CRAY-1 Computer System ». *Communications of the ACM*, 21(1) :63–72. Voir [Gajski et al., 1987].
- [Slotnick et al., 1962] Slotnick, D. L., Borck, W. C., and McReynolds, R. C. (1962). « The SOLOMON Computer ». In *Proceedings of the Fall 1962 Eastern Joint Computer Conference*, pages 97–107.
- [Szymanski, 1989] Szymanski, T. (1989). « On the permutation capability of a circuit-switched hypercube ». In *International Conference on Parallel Processing*, pages 103–110. The Institute of Electrical and Electronics Engineers, Inc., Academic Press.
- [Thornton, 1964] Thornton, J. E. (1964). « Parallel Operation in the Control Data 6600 ». In *Proceedings of the 1964 Fall Joint Computer Conference*, pages 33–40.
- [Tjaden and Flynn, 1970] Tjaden, G. S. and Flynn, M. J. (1970). « Detection and Parallel Execution of Independent Instructions ». *IEEE Transactions on Computers*, C-19(10) :889–895.
- [TMC, 1987] TMC (1987). « *Connection Machine Model CM-2 Technical Summary* ». Thinking Machine Corporation. HA87-4.
- [Tomasulo, 1967] Tomasulo, R. M. (1967). « An Efficient Algorithm for Exploiting Multiple Arithmetic Units ». *IBM Journal*, 11(1) :25–33.



- [Unger, 1958] Unger, S. H. (1958). « A Computer Oriented Toward Spatial Problems ». In *Proceedings of the IRE*, pages 1744–1750.
- [Wu and Feng, 1980] Wu, C.-L. and Feng, T.-Y. (1980). « On a Class of Multistage Interconnection Networks ». *IEEE*

Transactions on Computers, C-29(8) :694–702.

- [Wu and yun Feng, 1984] Wu, C.-L. and yun Feng, T. (1984). *Tutorial : Interconnection Networks for Parallel and Distributed Processing*. IEEE Computer Society Press.



- Liste des transparents
- 0 Titre : Cours d'Architecture des Ordinateurs « Parallèles »
 - 1 Introduction
 - Introduction**
 - 2 Quelques applications
 - 3 Tokamak — physique nucléaire
 - 4 Prévision météorologique
 - 5 Prévision des tornades
 - 7 Contraintes sur les ordinateurs
 - 8 Prix Gordon Bell
 - 12 Top 500 — 2006
 - Top 500**
 - 14 Le Top 10 — 2006
 - 16 Parallélisme massif — 2006
 - 17 Architectures — 2006
 - 19 Types de processeurs — 2006
 - 20 Performance globale
 - 21 Top 500 — 2003
 - 23 Plan
 - 24 Époque pré-électricité
 - Histoire**
 - 23 Électrification
 - 27 Architecture monoprocesseur classique
 - 33 Transistor
 - 34 Première génération
 - 35 Deuxième génération
 - 36 Troisième génération
 - 37 Microprocesseurs
 - 39 Intégration à grande échelle
 - 41 Résumé évolution vitesse
 - 42 Évolution technologique

- # Architecture
- 41 Définition de performance
 - 44 Ordinateur séquentiel
 - 45 Limitations du séquentiel
 - 46 Architecture Harvard
 - Parallélisme**
 - 47 Limitations du parallélisme
 - 50 Critères économiques...
 - 53 Critères économiques parallèles
 - 54 Travail de l'architecte
 - 55 Recibler un VAX...
 - 56 Mesure du parallélisme — quelques termes
 - 57 Loi d'Amdahl
 - 59 Discussion
 - 60 Efficacité en fonction du nombre de processeurs
 - 61 La superlinéarité existe-t-elle ?
 - 63 La sublinéarité existe...
 - 65 Approche psychologique
 - 66 Approche psychologique pour le //isme
 - 67 Mesure des performances
 - 69 Niveaux de parallélisme
 - 70 Quand paralléliser ?
 - 71 Base de la parallélisation
 - 73 Pourquoi le parallélisme ?
 - 74 Parallélisme intraprocesseur
 - Parallélisme intraprocesseur**
 - 75 Ordinateur typique
 - 76 Ordinateur typique minimal
 - 77 Pipeline
 - 79 Efficacité du pipeline

- 81 Pas si simple...
- 82 Les risques du pipelines
- 84 Mémoire monoport
- 85 Dépendances de données
- 86 Autres dépendances de données
- 87 Planification du code
- 89 Ordinateur vectoriel
- 90 μ VP (Fujitsu)
- 91 μ VP - Synoptique
- 92 μ VP - Unité vectorielle
- 93 μ VP - Multiplieur
- 94 μ VP - Additionneur
- 95 μ VP - Le circuit
- 96 Gestion des dépendances de données
- 97 Cas de pipeline explicite
- 98 Gestion des dépendances de contrôle
- 99 Ajout de matériel
- 100 Prédiction des branchements
- 101 Prédiction statique
- 102 Branchement retardé
- 103 Remplissage après un branchement retardé
- 104 Prédiction dynamique
- 105 Prédiction de branchement à 2 bits
- 106 Les exceptions
- 107 Solutions
- 108 Pipeline à exécution dynamique
- 109 Version simplifiée
- 110 Scoreboard
- 111 Algorithme de Tomasulo
- 113 Intérêt de l'algorithme de l'algorithme de Tomasulo
- 114 Renommage de registres
- 115 Déroulage des boucles
- 116 Pipeline logiciel

- 117 Processeur superscalaire
- 118 PentiumPro
- 119 PentiumPro — le circuit
- 120 Exécution spéculative
- 122 Faire du RISC avec du CISC ?
- 123 Intel Pentium 4
- 124 AMD Opteron
- 125 AMD Opteron
- 127 VLIW
- 128 VLIW – caractéristiques
- 129 Intel Itanium2
- 130 Dépendance de contrôle \rightsquigarrow données
- 131 Addition entière

Unités arithmétiques et logiques

- 132 Divertissement parallèle
- 133 Opération préfixe parallèle (scan)
- 136 Additionneur *carry-lookahead*
- 137 Multiplication entière
- 139 Multiplication par arbre de Wallace
- 140 Opération multiplication & addition
- 142 Quelques vitesses de microprocesseurs
- 146 Conclusion intraprocesseurale
- 147 Parallélisme interprocesseur

Parallélisme interprocesseur

- 148 Taxinomie de Flynn
- 152 SIMD

- 154 Le contrôle
- 155 Les processeurs élémentaires
- 156 MIMD
- 158 Synchronisation en MIMD
- 160 Taxinomie para-Flynnienne
- 161 Ordinateurs vectoriels
- 162 Vectoriel
- 163 Type de machines vectorielles
- 164 MISD
- 166 Et le SPMD ?
- 167 Nombre de processeurs ↵ Taille des processeurs
- 168 Gros grain
- 169 Grain fin
- 171 Grain moyen
- 172 Dépasser le « mur de la mémoire »

- 171 **La mémoire**
- 171 **Mémoire**
- 173 Hiérarchie mémoire
- 175 Mémoire DRAM type PC
- 176 Le cours de la bourse... euh... mémoire
- 178 Mémoires dynamiques (DRAM)
- 182 Mémoires modernes
- 185 Hiérarchie mémoire Origin2000
- 186 Mémoire banquée
- 188 Mode de couplage avec la mémoire
- 190 Couplage fort ≡ mémoire partagée
- 192 Couplage faible ≡ mémoire distribuée
- 194 Placement en mémoire
- 195 Le programmeur & la mémoire
- 196 SMP & AMP
- 197 Réseaux d'interconnexion

Les réseaux

- 196 Topologie
- 198 Stratégie de contrôle
- 200 Méthode de commutation
- 202 Synchronisme
- 203 Conception d'un réseau
- 204 Optimisation d'un réseau
- 206 Les réseaux statiques
- 207 Réseau totalement connecté
- 208 Hypercube
- 209 Grilles et tores
- 212 Graphes de De Bruijn et de Kautz
- 214 Cube-connected cycles
- 215 Réseaux plus exotiques
- 217 Réseau dynamique
- 218 Le bus
- 219 Réseau à commutateurs à croisillons
- 220 Réseaux de type Oméga
- 222 Réseau de Clos
- 224 Réseau de Benes
- 226 Autres réseaux dynamiques
- 227 Méthodes de commutation
- 228 Techniques de routage
- 230 Éviter les interbloquages
- 231 Infiniband

- 230 **Exemples de réseaux**
- 233 Quadrics
- 234 Myrinet
- 235 Ethernet
- 236 Topologie
- 237 Accélération des entrées-sorties

236 Les Entrées-Sorties

- 238 RAID
- 239 Redundant Array of Inexpensive Disks
- 240 Types de RAID
- 241 Les tendances actuelles

240 Les tendances actuelles

- 243 Considérations systèmes en matériel
- 244 Systèmes reconfigurables
- 245 Cray XD1
- 246 Cartes graphiques
- 248 Multiprocesseur mémoire partagée

247 Le Zoo Mémoire partagée

- 249 SPARC Server 2000E — 1995
- 251 Convex Exemplar SPP1 — 1993
- 252 Convex Exemplar SPP1 — 1993
- 253 SGI Origin2000 — 1998
- 254 Architecture Origin2000
- 255 SGI Origin3000 — 2001
- 259 TERA - MTA — 1996
- 260 Intel ASCI Red — 1995

259 Mémoire distribuée

- 261 IBM p5-575 — 2005

- 262 IBM Power 5
- 263 Bull Tera-10
- 264 IBM BlueGene/L
- 266 Tolérance aux pannes
- 267 NEC SX-4R — 1996

266 Vectoriels

- 271 NEC SX-6 — 2002
- 272 Nec SX-8 — 2005
- 273 Cray Inc. — la saga

272 Cray & co.

- 275 Cray Computer Corporation — CRAY 4
- 276 Gamme Cray
- 277 Cray SV1 — 1999
- 278 Cray T3E-1350 — 2000
- 279 Cray X1E — 2005
- 280 MasPar MP-2 — 1993

279 SIMD

- 282 Construire son supercalculateur...

281 Conclusion

- 283 Conclusion
- 286 *Real Politik*
- 287 Défis futurs
- 288 Bibliographie

288 *

289 *