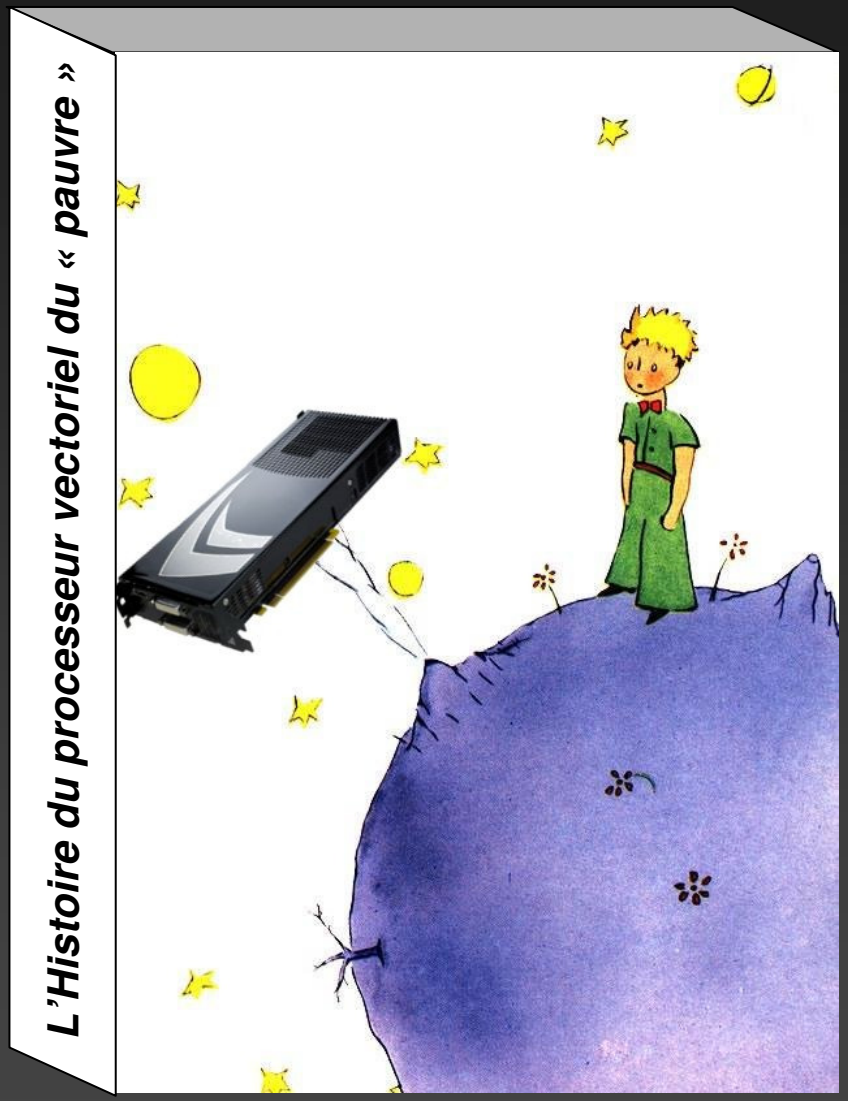
Dis… c'est quoi un GPU ?

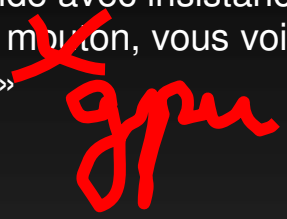
David Defour

Laboratoire ELIAUS,
Université de Perpignan



Préface

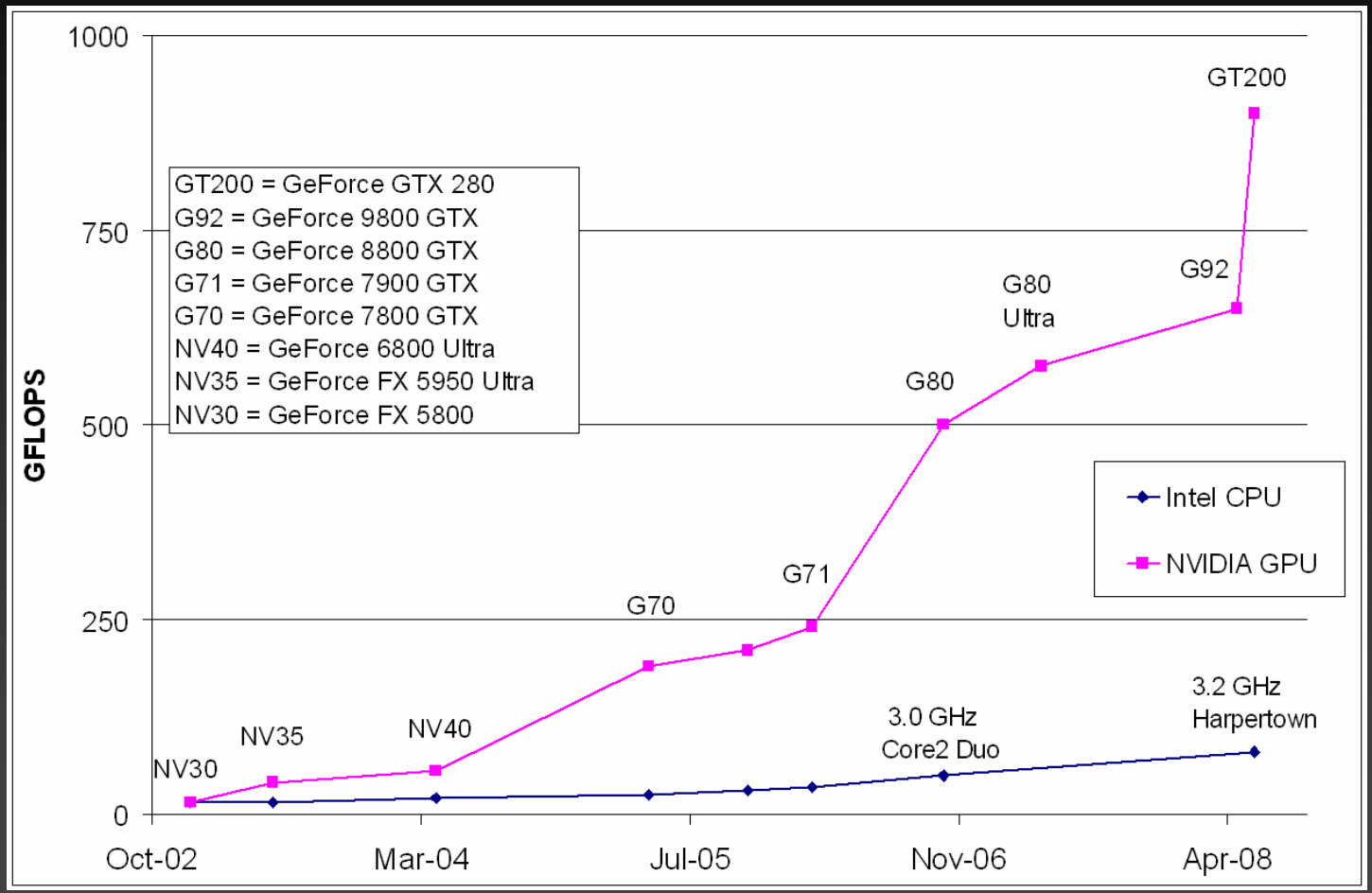
« Imaginez-vous perdu dans le désert, loin de tout lieu habité, et face à un petit garçon tout blond, surgi de nulle part. Si de surcroît ce petit garçon vous demande avec insistance de dessiner un mouton, vous voilà plus qu'étonné ! »

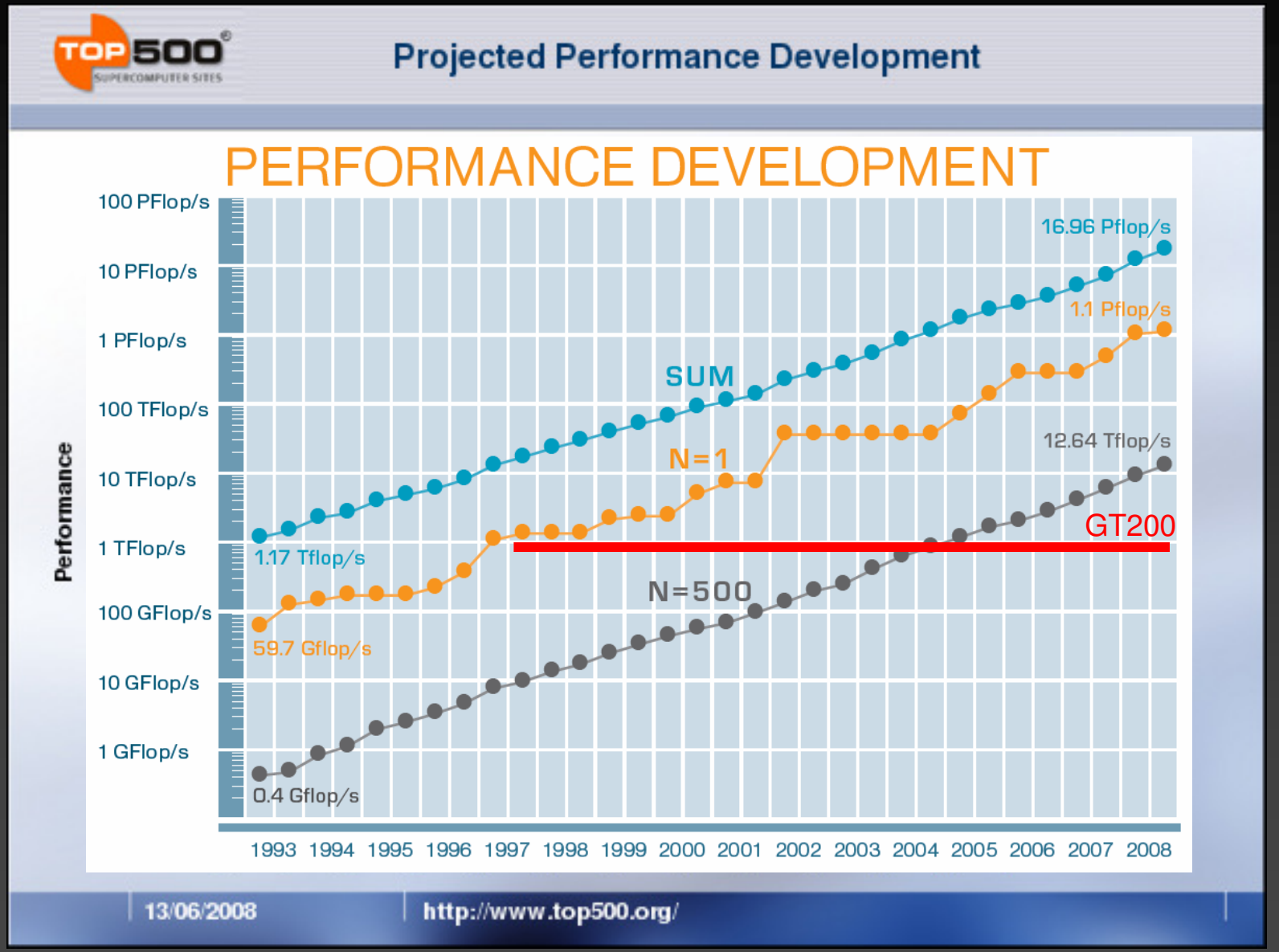


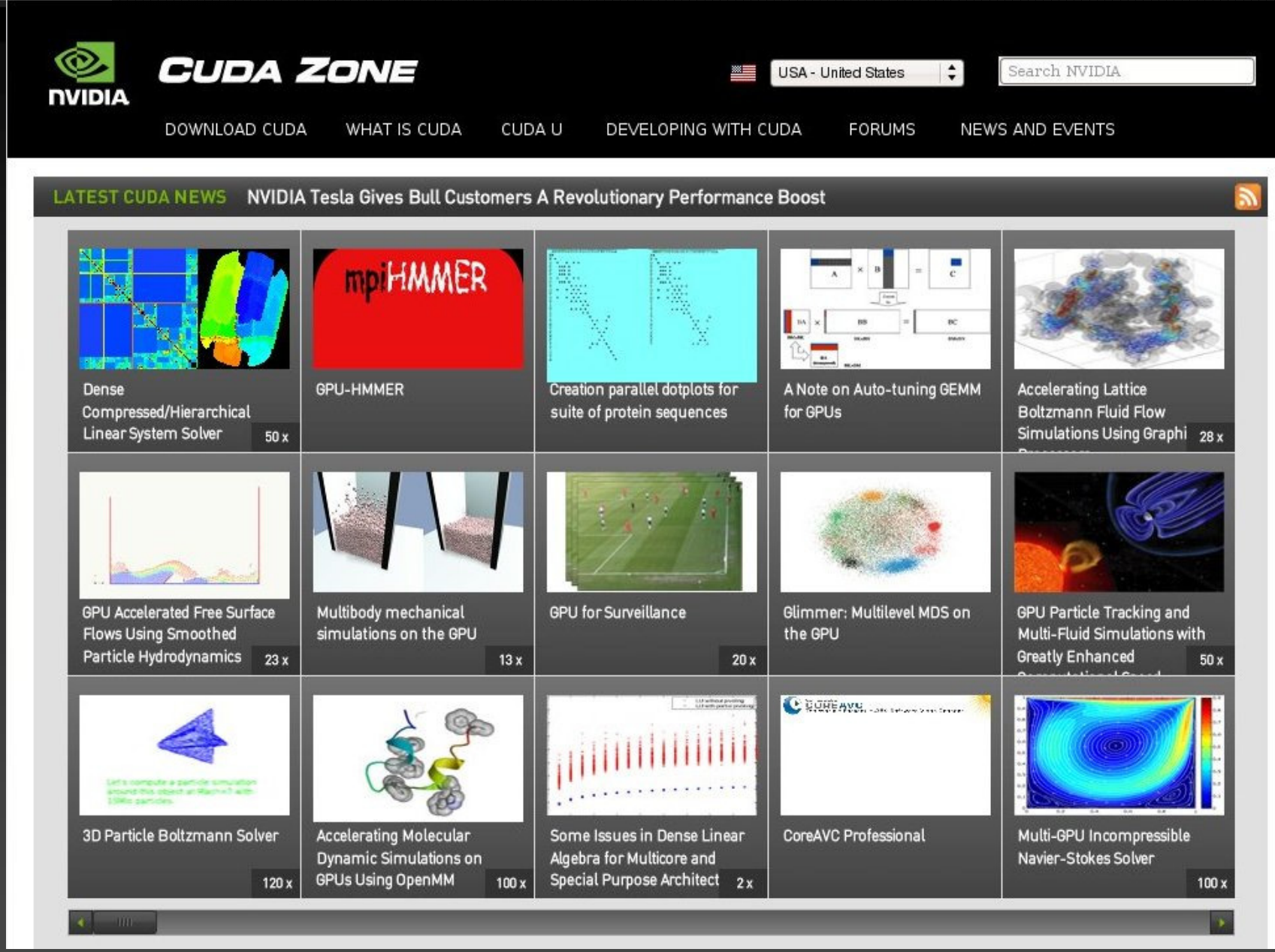
- Ce que l'on verra :
  - Les grandes lignes du calcul vectoriel
  - Implications sur le design interne
  - Interactions entre langage/compilateur/matériel
- Ce que l'on ne verra pas
  - Relation entre GPU et multicoeur
  - Programmation GPGPU, //, CUDA, HMPP…

Chapitre 1

GPU puissant (co)processeur vectoriel ?


1000
750
500
250
0
GFLOPS
GT200 = GeForce GTX 280
G92 = GeForce 9800 GTX
G80 = GeForce 8800 GTX
G71 = GeForce 7900 GTX
G70 = GeForce 7800 GTX
NV40 = GeForce 6800 Ultra
NV35 = GeForce FX 5950 Ultra
NV30 = GeForce FX 5800
GT200
G92
G80 Ultra
G80
G71
G70
NV40
NV35
NV30
3.0 GHz Core2 Duo
3.2 GHz Harpertown
Intel CPU
NVIDIA GPU
Oct-02
Mar-04
Jul-05
Nov-06
Apr-08


TOP500
SUPERCOMPUTER SITES
Projected Performance Development
PERFORMANCE DEVELOPMENT
Performance
100 PFlop/s
10 PFlop/s
1 PFlop/s
100 TFlop/s
10 TFlop/s
1 TFlop/s
100 GFlop/s
10 GFlop/s
1 GFlop/s
16.96 Pflop/s
1.1 Pflop/s
SUM
N=1
12.64 Tflop/s
GT200
1.17 Tflop/s
N=500
59.7 Gflop/s
0.4 Gflop/s
1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008
13/06/2008
http://www.top500.org/

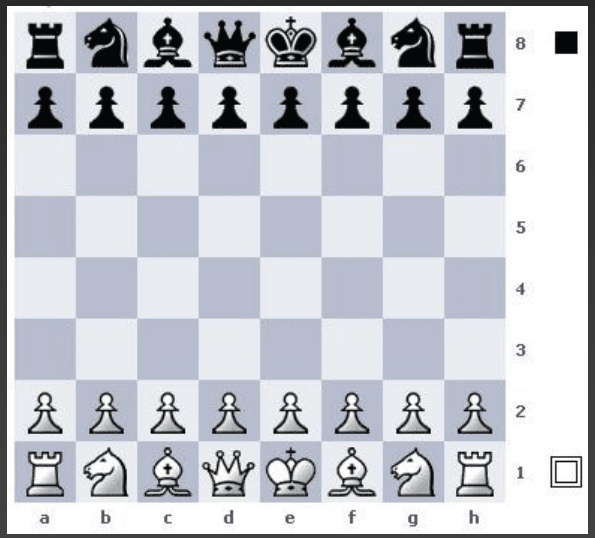

NVIDIA
CUDA ZONE
USA - United States
Search NVIDIA
DOWNLOAD CUDA
WHAT IS CUDA
CUDA U
DEVELOPING WITH CUDA
FORUMS
NEWS AND EVENTS
LATEST CUDA NEWS
NVIDIA Tesla Gives Bull Customers A Revolutionary Performance Boost
Dense Compressed/Hierarchical Linear System Solver
50 x
mpiHMMER
GPU-HMMER
Creation parallel dotplots for suite of protein sequences
A Note on Auto-tuning GEMM for GPUs
Accelerating Lattice Boltzmann Fluid Flow Simulations Using Graphi
28 x
GPU Accelerated Free Surface Flows Using Smoothed Particle Hydrodynamics
23 x
Multibody mechanical simulations on the GPU
13 x
GPU for Surveillance
20 x
Glimmer: Multilevel MDS on the GPU
GPU Particle Tracking and Multi-Fluid Simulations with Greatly Enhanced
50 x
3D Particle Boltzmann Solver
120 x
Accelerating Molecular Dynamic Simulations on GPUs Using OpenMM
100 x
Some Issues in Dense Linear Algebra for Multicore and Special Purpose Architect
2 x
CoreAVC Professional
Multi-GPU Incompressible Navier-Stokes Solver
100 x

GPGPU = Buzzword ?

- GP GPU =
  - Économique => Je peux l'acheter
  - Production de masse => Modèle pérenne
  - Vectorielle => J'aurais de la puissance
  - Accessible => Je peux programmer

Localité des données

- On s'est rendu compte très tôt qu'il existe des problèmes où l'on peut exploiter la localité spatiale des données…



• A.L. Samuel « Computing bit by bit », proc. IRE, vol 41, pp1223-1230, October 1953
• K.E. Iverson « APL, A Programming Language », 1957

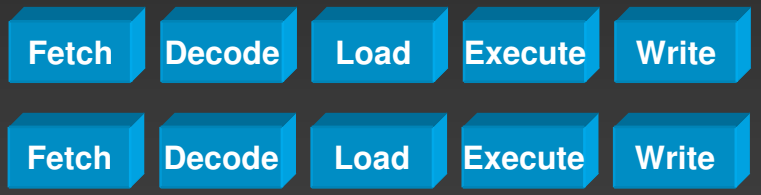
Ancêtres des GPU

- 1960 : 1ère apparition à Westinghouse
  - Projet Solomon, plusieurs ALU contrôlés par 1 CPU maître
- 1962 : projet Solomon abandonné
- 1972 : Projet repris à l'Université de l'Illinois
  - Naissance de l'ILLIAC IV (64 ALU pour 100 MFLOPS)
- 1974 : utilisation commerciale du CDC STAR-100
  - Introduction du pipeline d'instruction
  - Long pipeline pour accéder à la mémoire lente
  - Coût de changement important
  - Résultat décevant sur des cas réels….

Instruction Level Parallelism

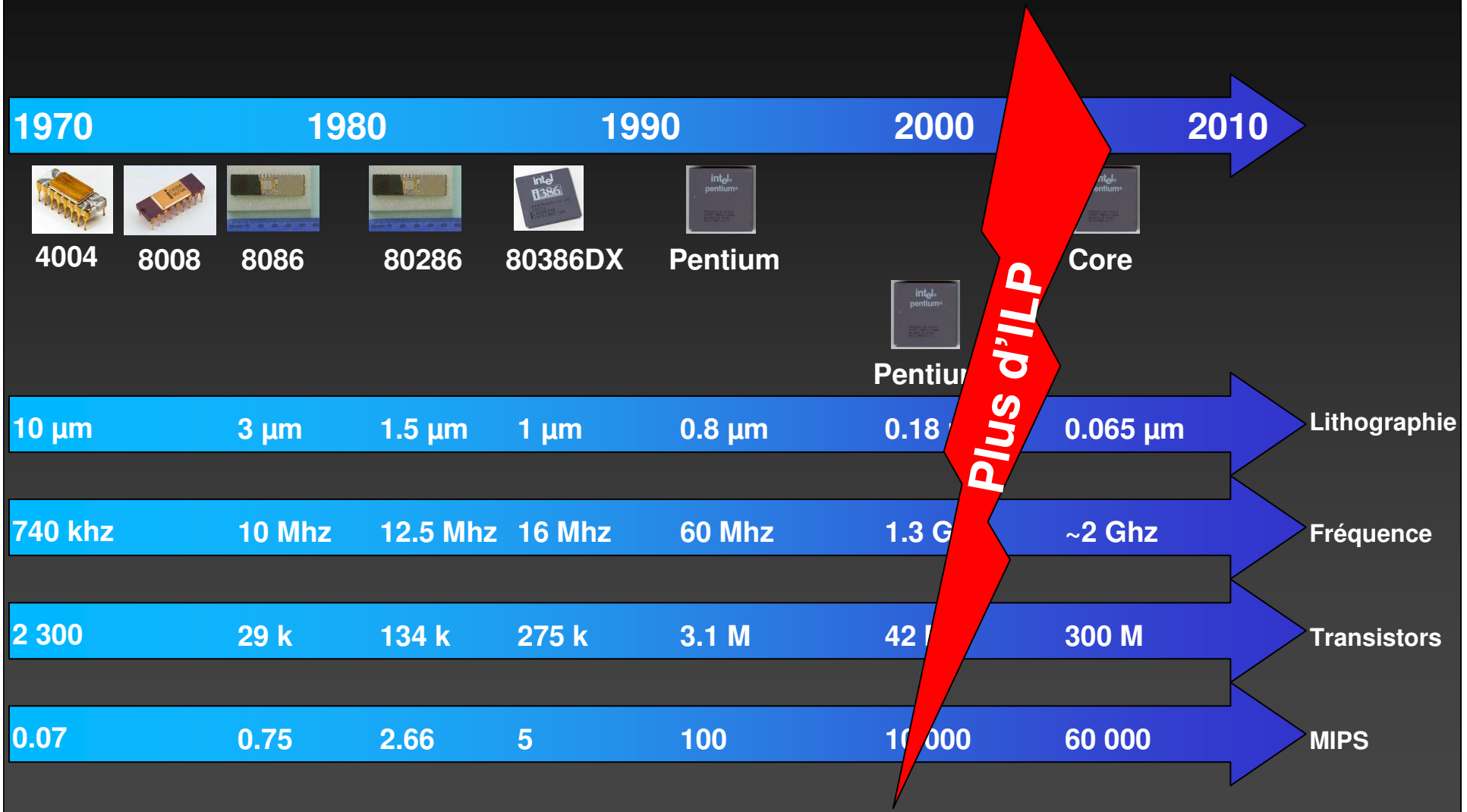
- Exécution pipelinée

- Exécution superscalaire



- Exécution out-of-order

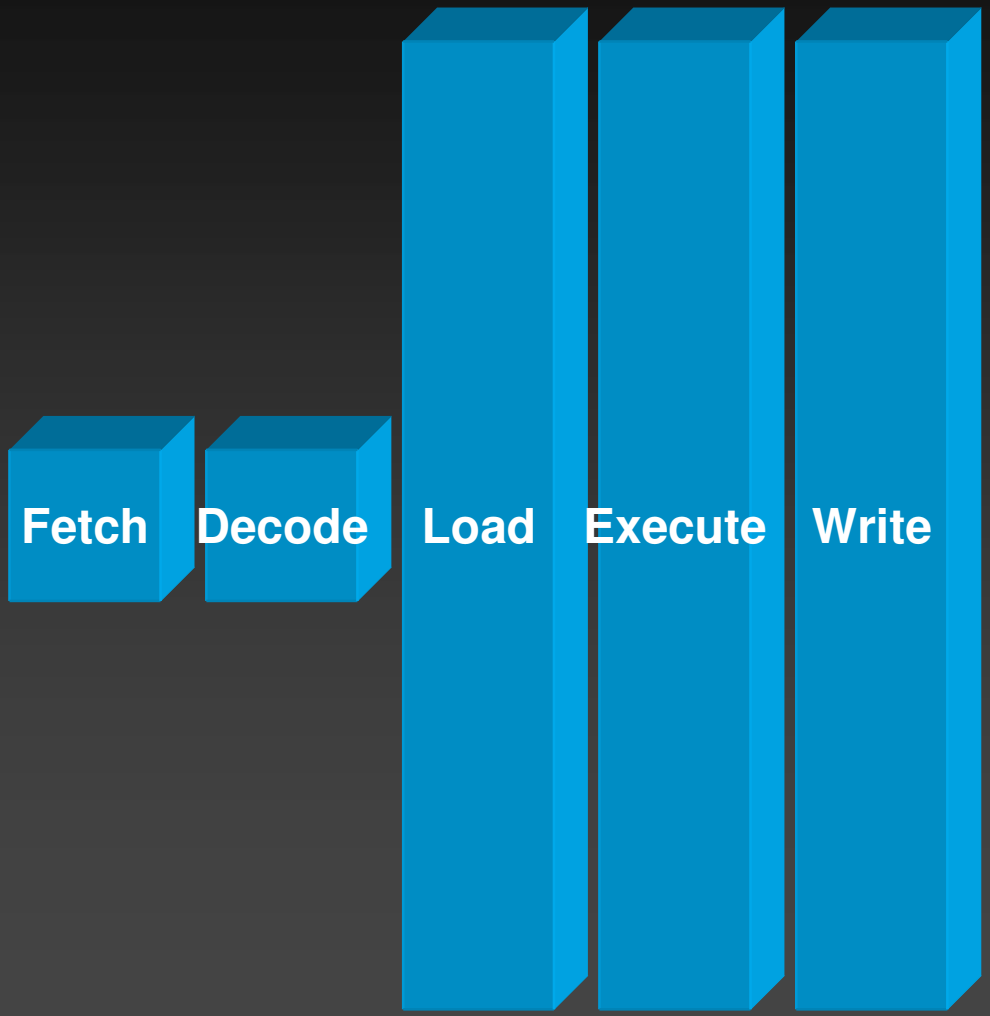
Processeurs Intel



Problèmes liés à l'approche conventionnelle

- Augmentation de la fréquence d'horloge
  - Augmente le # de cycles par instruction (Pb des branchements)
- Chargement & Décodage
  - Difficile d'en réaliser beaucoup + par cycle
- Taux de succès des caches
  - Dépend de l'application et mauvais sur :
    - les appli scientifiques (gros jeux de données)
    - Multimédia (traitement de flux)

Solution : la vectorisation du travail



Avantages du mode vectoriel

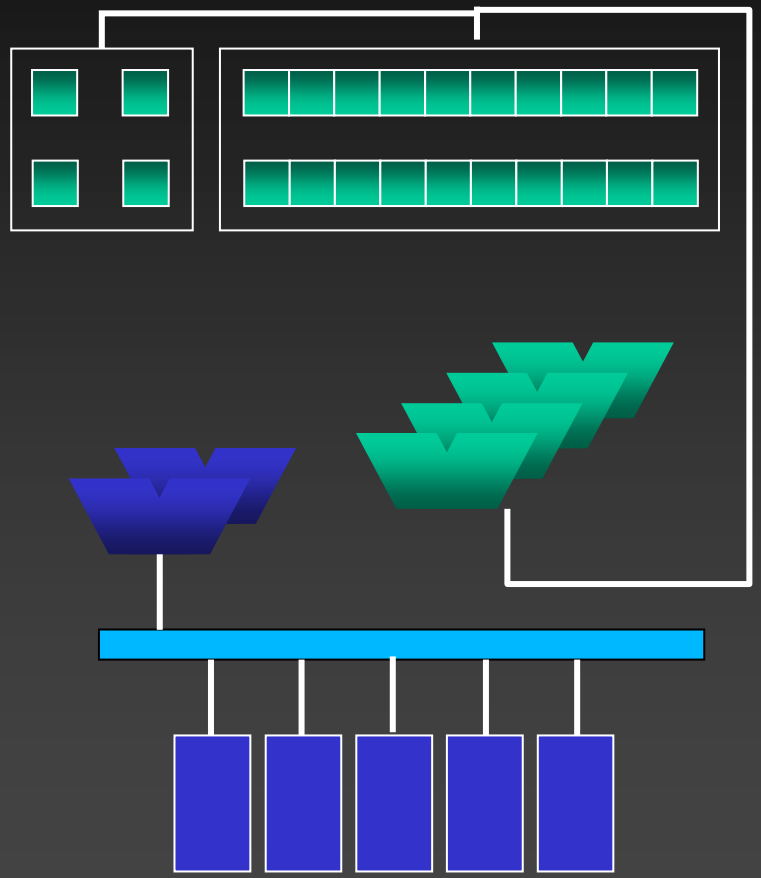
- Chaque résultat est indépendant du précédent
  - ➔ Long pipeline, avec compilateur qui gère les dépendances
  - ➔ Fréquence d'horloge élevée
- Les instructions vectorielles accèdent à la mémoire avec des motifs connus
  - ➔ Mémoire fortement entrelacée
  - ➔ Amortie les latences d'accès à la mémoire
  - ➔ Pas besoin de cache (sauf instruction !)
- Réduit les problèmes de dépendances
- Instructions vectorielles
  - ➔ Réduit le nombre de chargement d'instruction

Chapitre 2

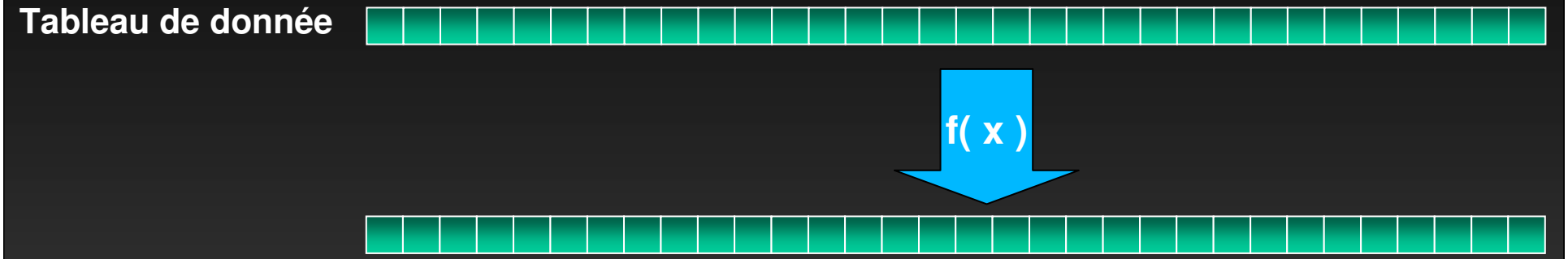
Processeur vectoriel minimaliste

Composant d'un processeur vectoriel

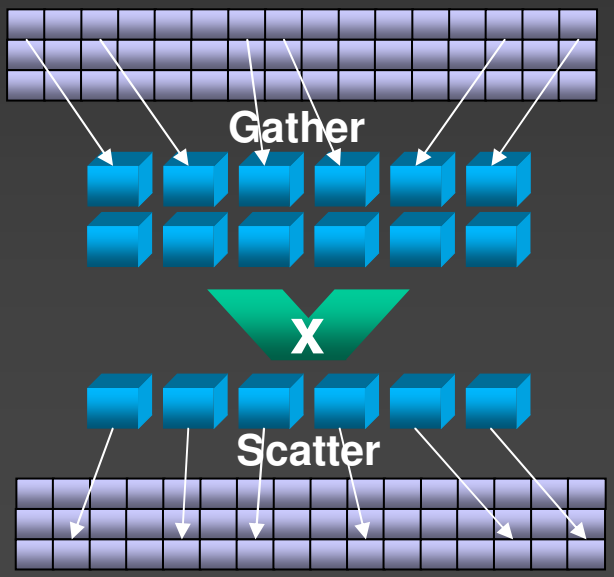
1. Registres vectoriels
   - Banc de longueur fixe
2. Unités vectorielles
   - Pipelinées, démarrent un nouveau calcul à chaque cycle
3. Unités Load/Store vectorielles
   - Pipelinées, possibilités de traiter plusieurs vecteurs
4. Registres scalaires
   - Utiles pour les opérations scalaires ou les adresses
5. Mémoire entrelacée
6. Bus de connexion



Terminologie



- Données contiguës en mémoire sur lesquelles on applique une fonction
  - Éléments :
    - Opérations arithmétiques (+/- de 2 vecteurs, …)
    - Définit par l'utilisateur
  - Communications :
    - Réduction, scan (additions des éléments d'un vecteur)
  - Permutations :
    - Scatter (A[i]=B), gather (A=B[i])
    - Pack/unpack, shift/rotate



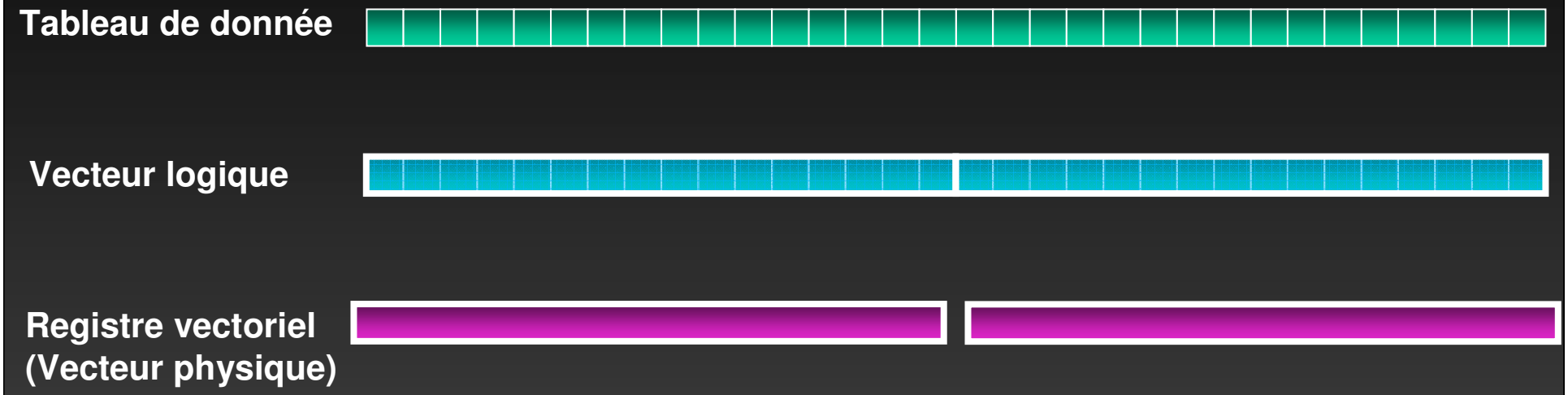
Terminologie

Tableau de donnée

Vecteur logique

Le tableau de donnée est découpé en vecteur afin de diviser le travail pour être traité en matériel.

Terminologie



- Banc de registres vectoriel
  - Chaque registre est un tableau d'élément
  - La taille des registres détermine la taille du vecteur pour une opération spécifique
- On dispose de plusieurs pipelines d'exécution

Taille du vecteur

- Comment gérer la taille du vecteur si != de la taille architecturale ?
- Registre de contrôle dédié au contrôle de chaque opération vectorielle (L/S ou opération)
  - ne peut pas être supérieur à la taille du registre architecturale
- Si taille du vecteur > taille max du vecteur
  - Découpage du travail

Terminologie

Tableau de donnée

Vecteur logique

Registre vectoriel (Vecteur physique)

Pipeline d'exécution

Chaque vecteur logique est traité dans un pipeline d'exécution propre.

Terminologie

Tableau de donnée

Vecteur logique

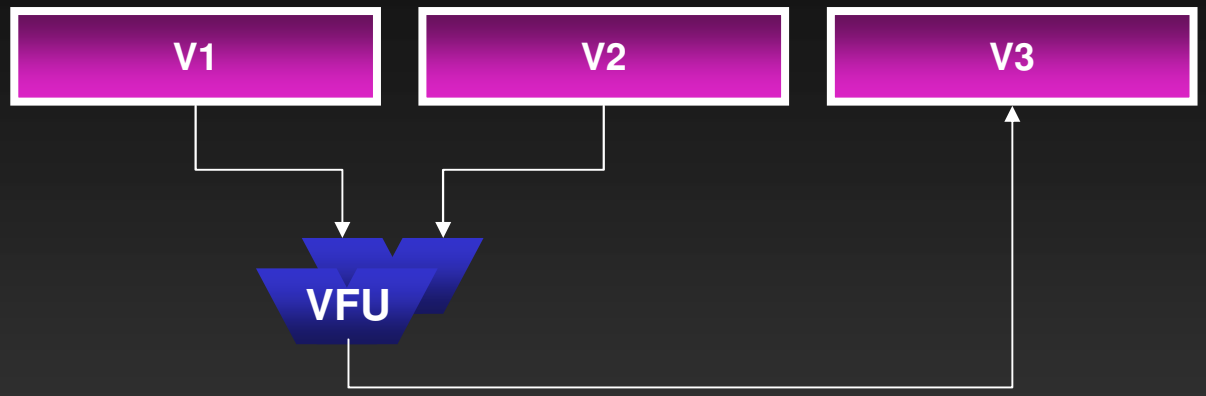
Registre vectoriel
(Vecteur physique)

Pipeline d'exécution

Exécuté par des VFU
(Vector Functional Unit)

Le traitement est réalisé par les unités vectorielles (VFU).

Modèle d'exécution



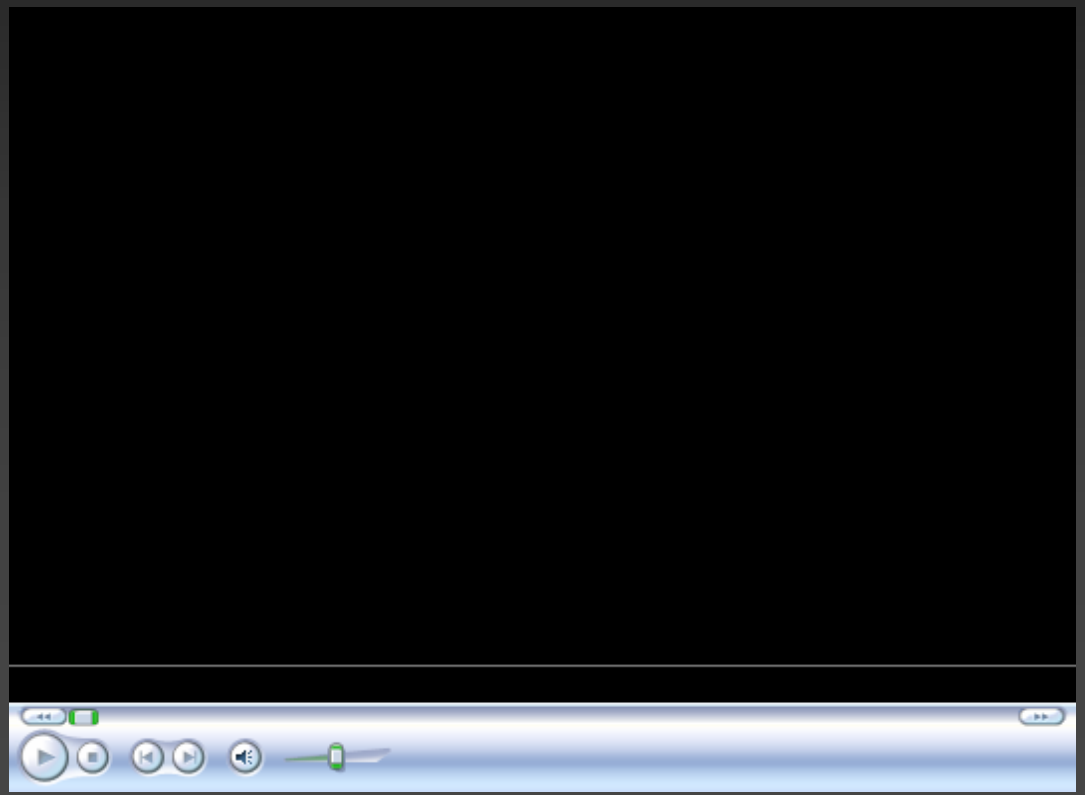
Registres vectoriels

MULLV V3 V2 V1

- Temps pour exécuter 1 inst.
  - = (Taille vecteur / taille de la VFU) * débit VFU

Qui veut gagner de l'argent en masse ?

- **Question :** « Comment intégrer les contraintes matérielles actuelles dans ce processeur ? »
- **Ma réponse :** JOKER… j'appel un spécialiste



Dave Patterson
Professeur à UC Berkeley

19:50 – 23:00

Chapitre 3

Optimisations vectorielles

Optimisations vectorielles

1. Gestions des registres
   1. Le chaînage
   2. Le registre de bypass
   3. Vecteur logique, vecteur physique
   4. Les registres partagées
2. Gestion de l'exécution
   1. Le multitâche
   2. L'exécution conditionnelle
   3. Gestion des aléas
3. Gestion de la mémoire
   1. Mémoire organisée en banc
   2. Load/store groupés
4. Gestion de la communication
   1. Communication inter-process
   2. Les bus

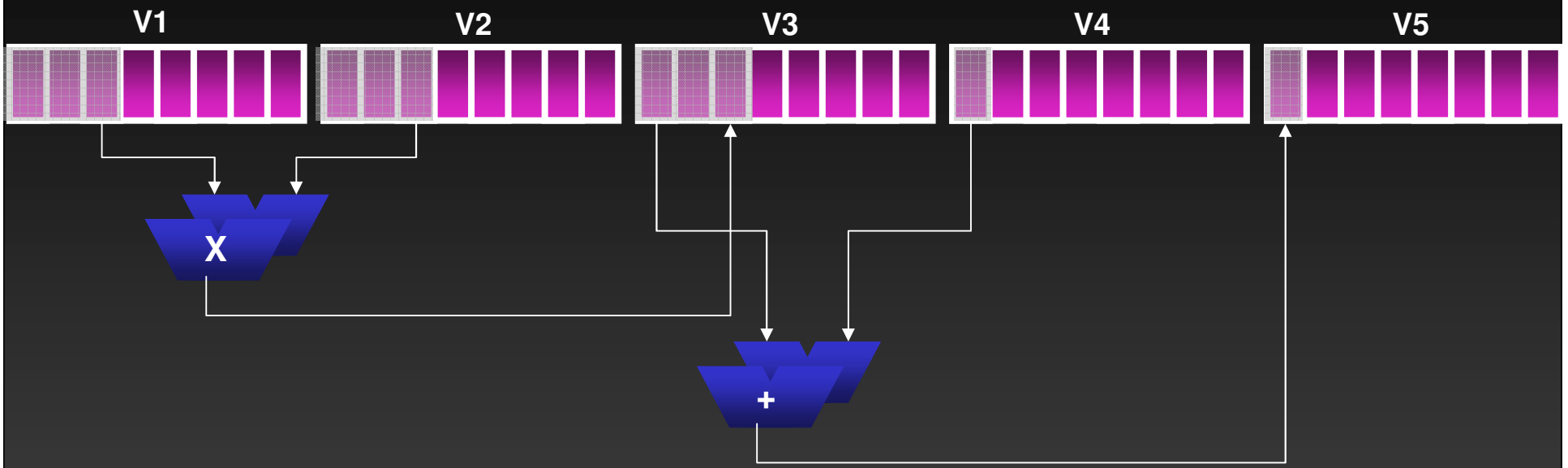
Temps de démarrage

- Pourquoi ne pas recouvrir le temps de démarrage d'une instruction vectorielle avec la suivante ?
  - Machine CRAY (+ puce ECL) construite sur ce principe

Gestions des registres, opt N°1

- Le chaînage
  - Supposons
    MULV V3, V2, V1
    ADDV V4, V3, V5
  - Le registre vectoriel V3 n'est pas vu comme un unique registre mais comme un groupe de registre individuel (on travail au niveau des éléments du vecteurs)
- Chainage flexible autorise un vecteur à se connecter à n'importe quel opération vectorielle active
- ☺ Augmente le nb d'instruction exécutable en vol.
- ☹ Plus de port en R/W

Chaînage



MULLV V3 V2 V1

ADDV V4 V3 V5

- Pipeliner les instructions qui se suivent fréquemment dans le code source

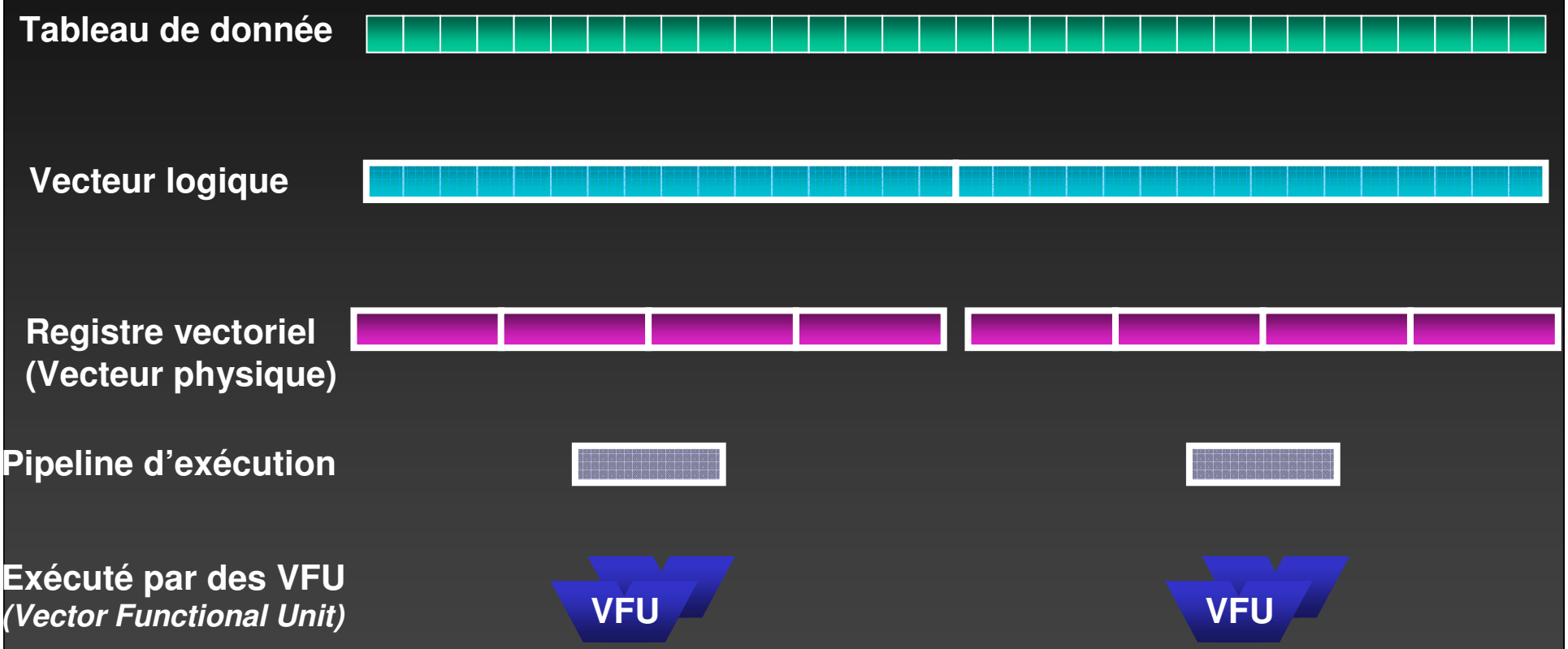
Gestions des registres, opt N°2

- Registre de bypass
- Ne pas utiliser un registre dédié pour un résultat consommé immédiatement après
  - Semblable au bypass matériel des superscalaires
  - Accessible au niveau architectural (ISA)

Gestions des registres, opt N°3

- Taille vecteur logique ≠ Taille vecteur physique
- Niveau d'abstraction supplémentaire :
  - On utilise plusieurs vecteurs physiques (registres vectorielles) pour représenter le vecteur « logique ».
  - Permet de recouvrir le temps de démarrage et les latences de traitement
  - Pas d'augmentation du nb de ports mais du nb de registres adressables
  - Maximisation des performances lorsque :
    Taille vecteur = nb max registre / nb registre pour l'exécution de la fonction.

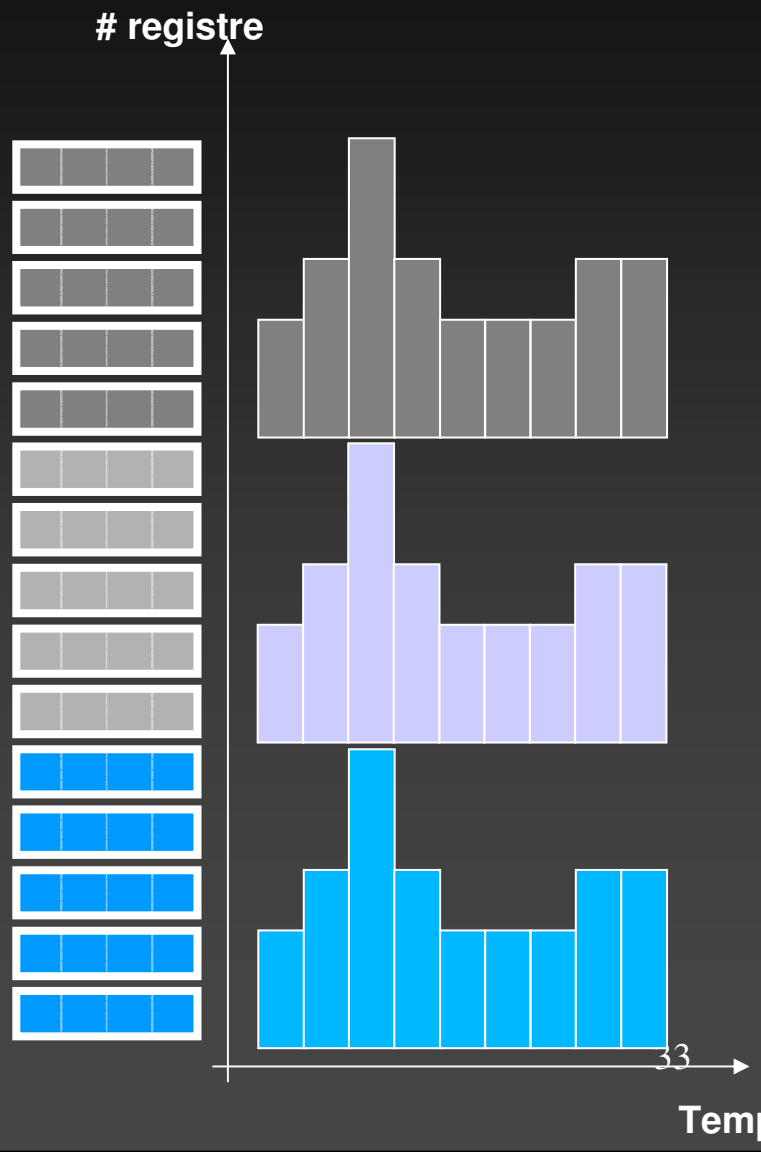
Gestions des registres, opt N°3



taille des registres ≠ taille du vecteur ≠ taille des VFU

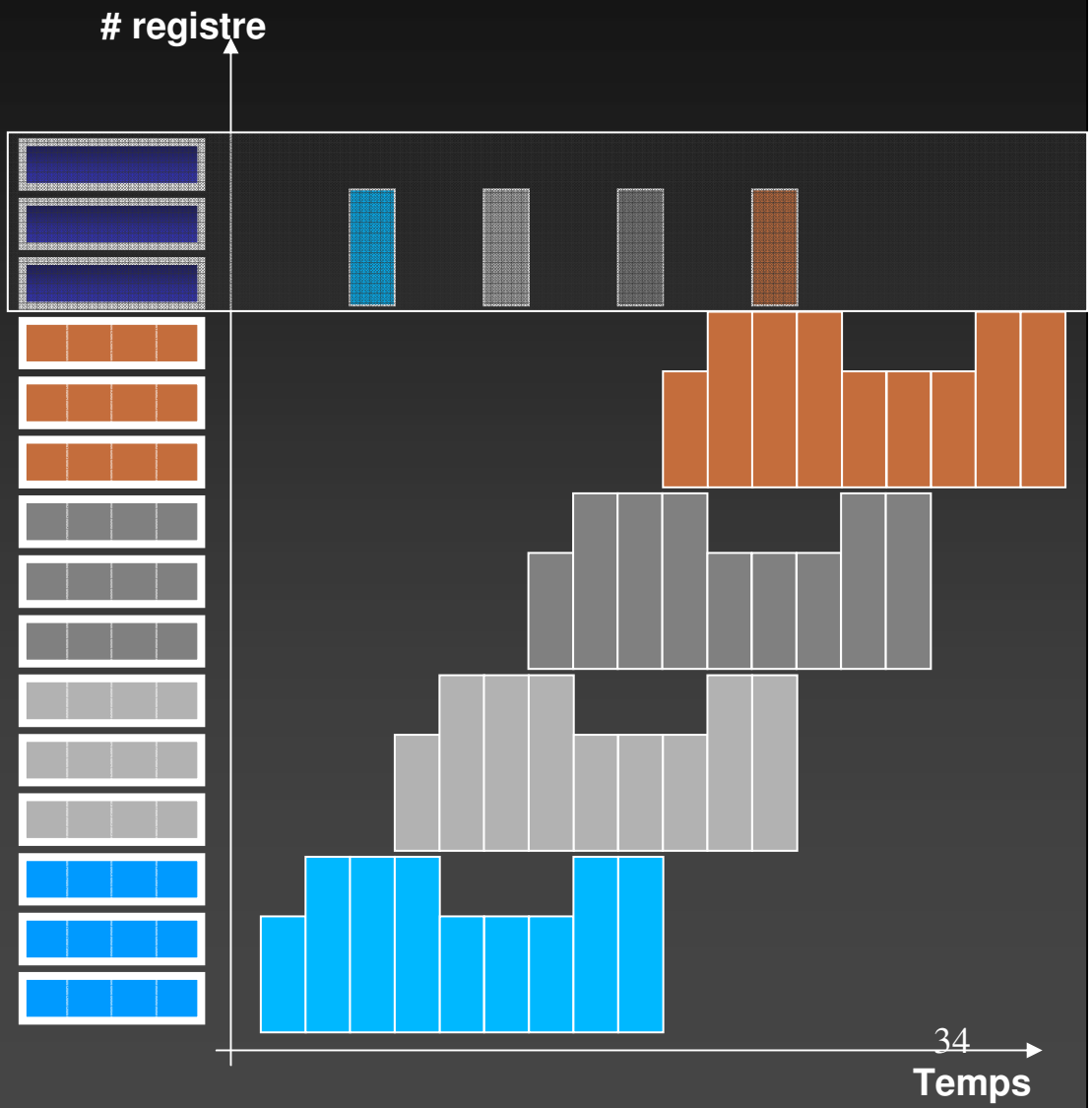
Gestions des registres, opt N°4

- Registres partagés
- Décaler l'exécution des vecteurs pour factoriser leur utilisation



Gestions des registres, opt N°4

- Registres partagés
- Décaler l'exécution des vecteurs pour factoriser leur utilisation



Optimisations vectorielles

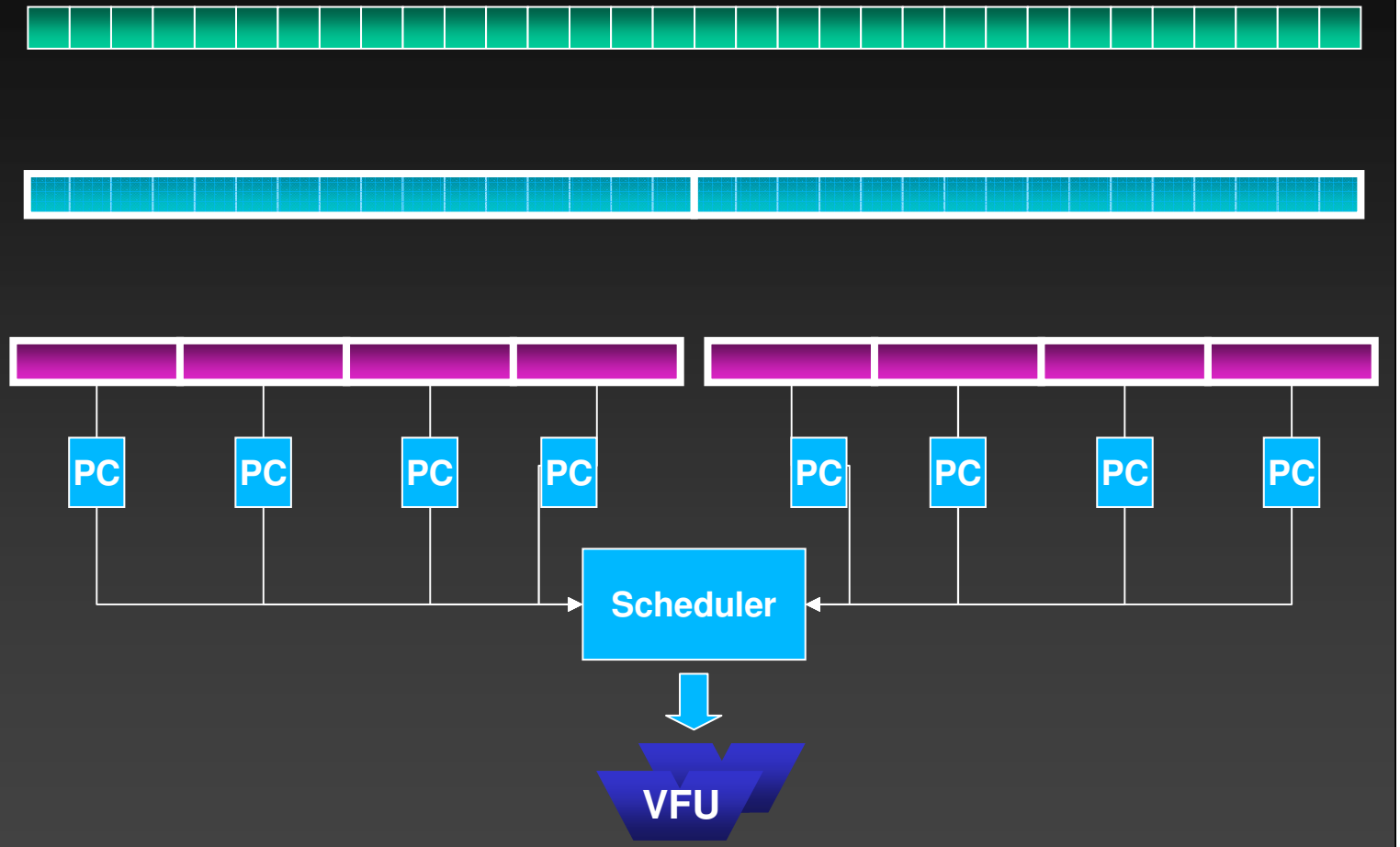
1. Gestions des registres
    1. Le chaînage
    2. Le registre de bypass
    3. Vecteur logique, vecteur physique
    4. Les registres partagées
2. Gestion de l'exécution
    1. Le multitâche
    2. L'exécution conditionnelle
    3. Gestion des aléas
3. Gestion de la mémoire
    1. Mémoire organisée en banc
    2. Load/store groupés
4. Gestion de la communication
    1. Communication inter-process
    2. Les bus

Gestion de l'exécution, opt N°1

- Multi-tâches

- On utilise les ressources disponibles pour autoriser le traitement de plusieurs vecteurs logiques au sein d'une même ligne.
- Permet de recouvrir le temps de démarrage et les latences de traitement
- Augmente le taux d'occupation des UF
- Nécessite un scheduler et un PC pour traiter chaque vecteur physique.

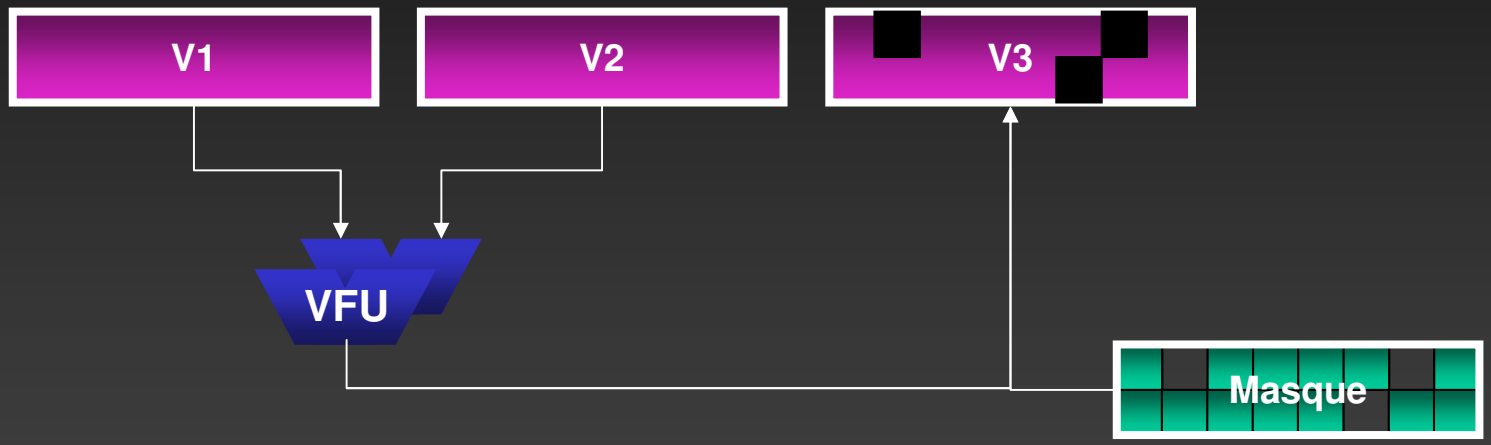
Multi-tâches



Gestion de l'exécution, opt N°2

- Exécution conditionnelle
- Vecteur de masque de contrôle: prend un vecteur de booléen pour contrôler le comportement des éléments d'un vecteurs
- Nécessite des cycles d'horloge, même pour ne rien faire !!!

Exécution conditionnelle



Gestion de l'exécution, opt N°3

- Et la gestion du alea, comme sur les CPU ?
  - Pas de mécanisme de pile
    - Pas de gestion des appels récursifs
  - Pas de cache (sauf inst., text, const.)
    - Bien pour WCET.
  - Pas d'Out-Of-Order
    - Moins de matériel
- Qu'est ce qui ne coûte pas cher
  - Prédication des instructions
  - Registre de masque
  - Niveau : vecteur physique

Optimisations vectorielles

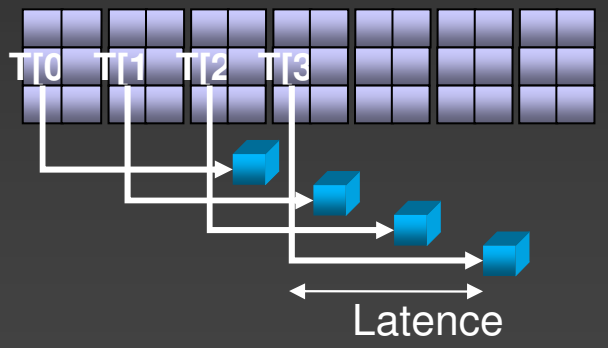
1. Gestions des registres
    1. Le chaînage
    2. Le registre de bypass
    3. Vecteur logique, vecteur physique
    4. Les registres partagées
2. Gestion de l'exécution
    1. Le multitâche
    2. L'exécution conditionnelle
    3. Gestion des aléas
3. Gestion de la mémoire
    1. Mémoire organisée en banc
    2. Load/store groupés
4. Gestion de la communication
    1. Communication inter-process
    2. Les bus

Gestion de la mémoire, opt N°1

- Unité mémoire vectorielle

- Temps de démarrage plus long que pour les UF
- La hiérarchie mémoire doit fournir (#ligne x taille vecteur) / cycle

- Utilisation de bancs entrelacés
  - 1) gère plusieurs L/S par cycle
  - 2) gère les accès non séquentiels
- Nb de bancs ~ taille vecteur physique



Gestion de la mémoire, opt N°2

- Vecteurs creux (stride)

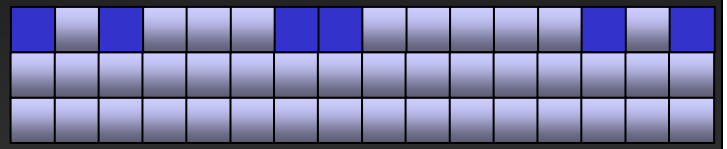
- Gestion des éléments non adjacents en mémoire
- Stride : distance séparant 2 éléments qui doivent être fusionné dans un seul vecteur (cache = stride de 1)
- Source de conflit mémoire
  (ex: stride de 32 sur 16 bancs)
- Nécessite de penser aux adresses des éléments vectoriels lors de la programmation

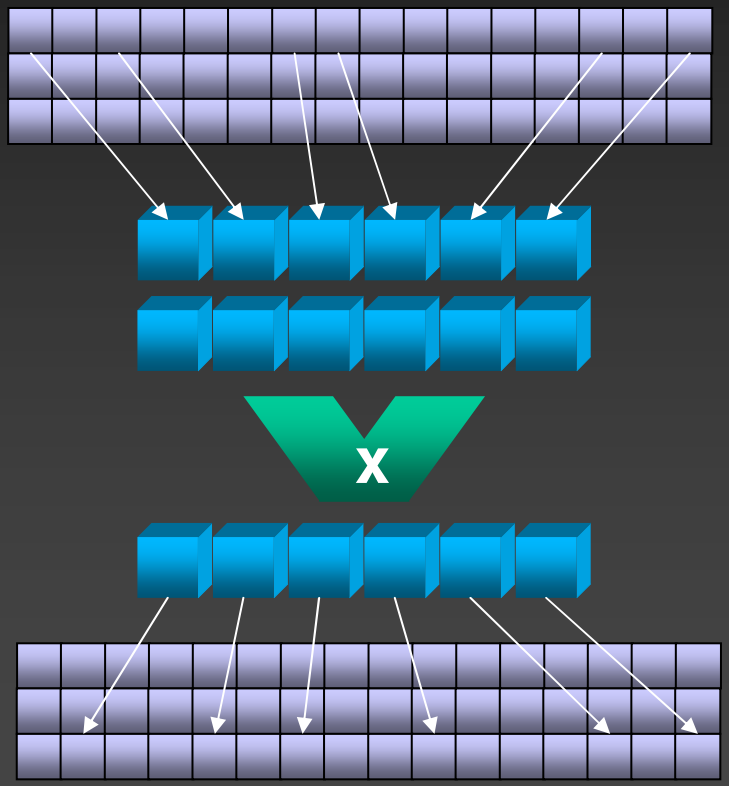
Gestion de la mémoire, opt N°3

- L / S de groupe de données entre les registres et la mémoire
  - Type d'adressage :
    - Stride unitaire
      - Le + facile/rapide
    - Stride constant (!=1)
    - Indexé (gather-scatter)
      - Équivalent vectoriel de l'indirection de registre
      - Permet les représentations creuses
      - Augmente le nombre de programme vectorisable

Gather/Scatter

- 2 Solutions :
  - Masques





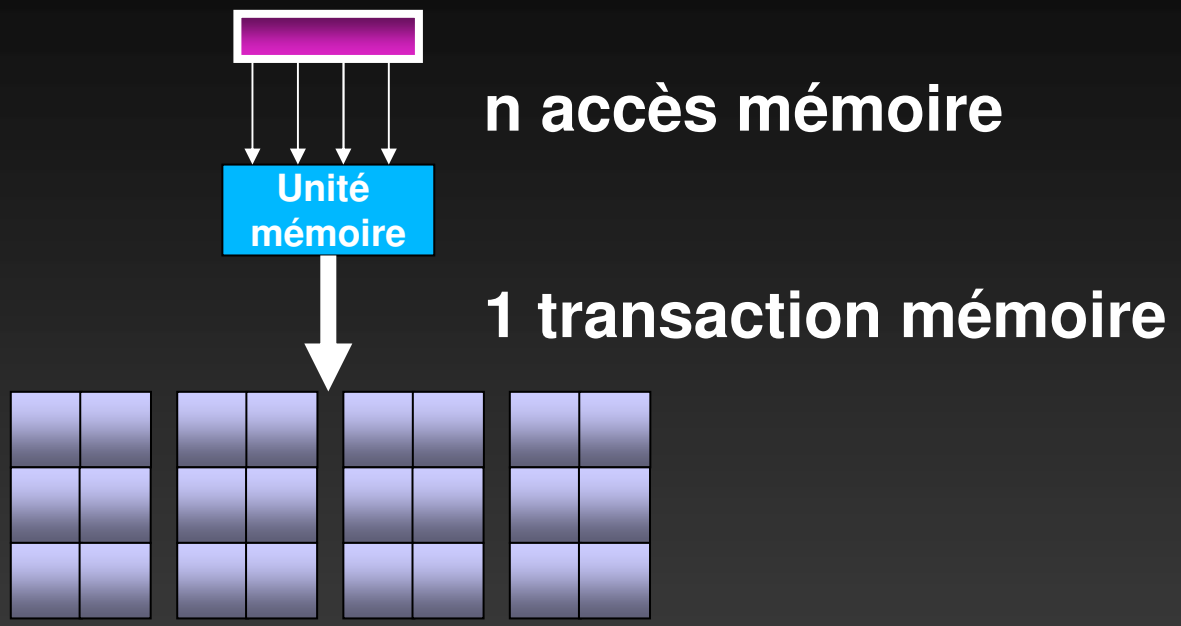
  - d'adresses

Gestion de la mémoire, opt N°3

- Accès mémoires groupés (mode burst)

- 1 instruction mémoire = 32 accès
- Utilisation d'un « buffer » pour traiter les différents motifs d'accès mémoire

Contraintes sur les accès mémoire



- n accès = 1 seule transaction si
  - Les adresses sont consécutives (stride de 1)
  - Les adresses ont un stride constant != du nombre de banc
- Tous les cas contraires
  - n transactions

Optimisations vectorielles

1. Gestions des registres
    1. Le chaînage
    2. Le registre de bypass
    3. Vecteur logique, vecteur physique
    4. Les registres partagées
2. Gestion de l'exécution
    1. Le multitâche
    2. L'exécution conditionnelle
    3. Gestion des aléas
3. Gestion de la mémoire
    1. Mémoire organisée en banc
    2. Load/store groupés
4. Gestion de la communication
    1. Communication inter-process
    2. Les bus

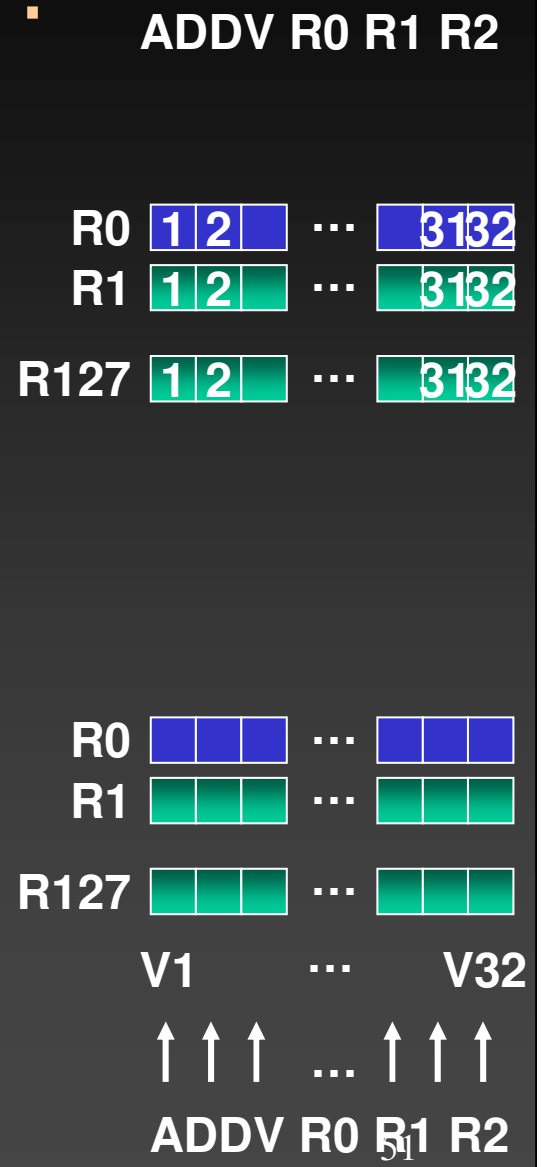
Besoin de communication

- Modèle vectoriel
  - A priori aucun processeur virtuel ne communique avec son voisin !
  - Pas besoin de processus de communication
- Sauf que…
  - La mémoire globale est lente
  - Algorithmes basés sur la communication (réduction…)
  - Mise à disposition d'une mémoire partagée
    - Plus barrière de synchronisation
    - Niveau le + intéressant : vecteur logique

Les bus

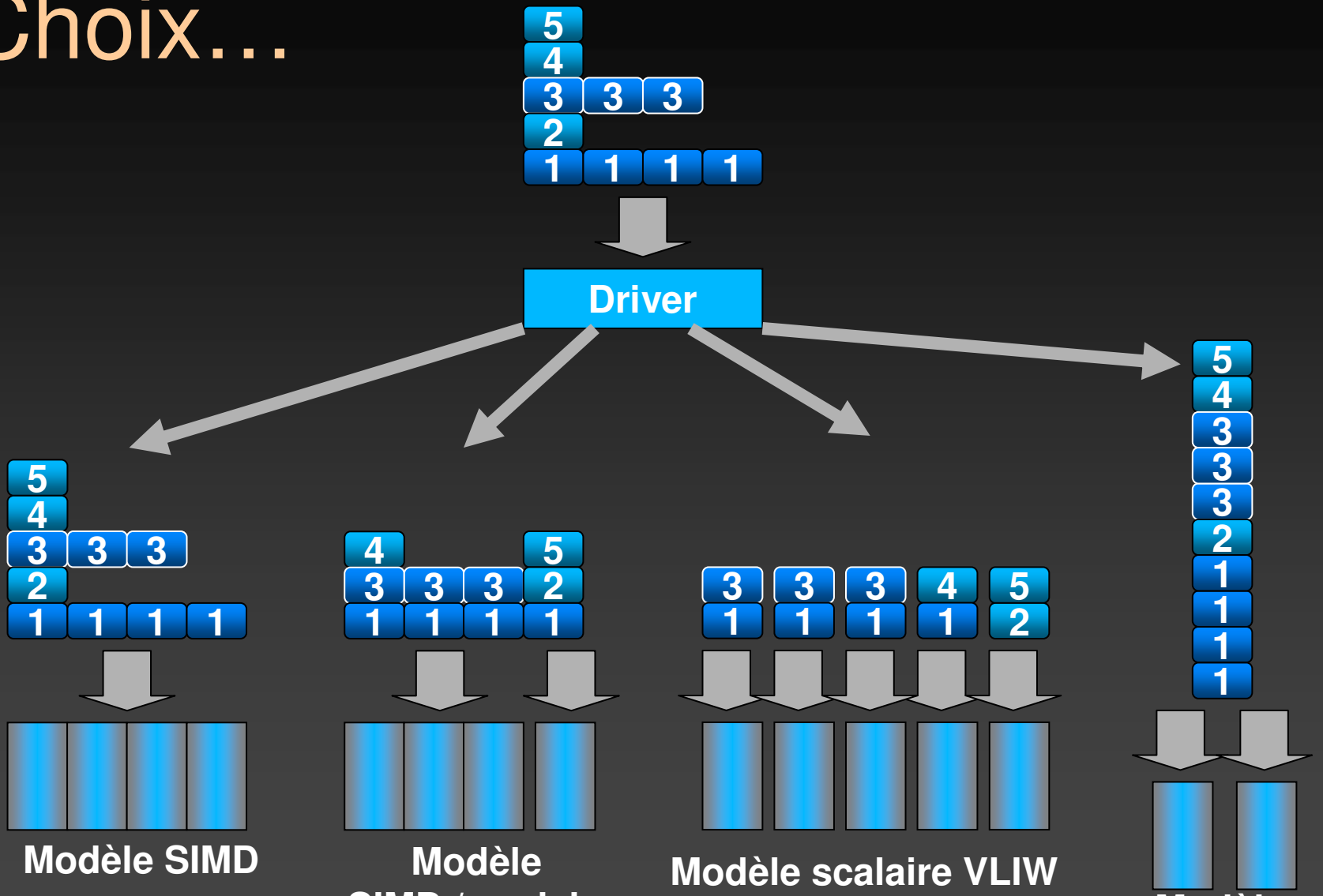
- Le bus de transfert des données
- Plusieurs solutions:
  - Bus
    (simple, mais peu performant)
  - Ring bus
    (coûteux mais bien si communication entre vecteur logique)
  - Crossbar
    (complexité quadratique en le nombre de client)

Quelle vision vectorielle ?



- Utilisation du vecteur à l'intérieur de la boucle
  - 128 Registres de 32 éléments
  - 1 instruction met à jour les 32 éléments d'un registre vectoriel
- Utilisation virtualisée du vecteur
  - 32 processeurs virtuels composés de 128 registres scalaires
  - 1 instruction met à jour 1 registre scalaire dans 32 processeurs virtuels
- Conceptuellement différent, Matériellement identique,

Choix…



Antiquité

ATI : R3xx - R5xx
Nvidia : NV4x - G7x

ATI : R6xx - R7xx

Nvidia : G8x - G9x - GT2xx

Chapitre 4

Quels rapports avec les GPU ?

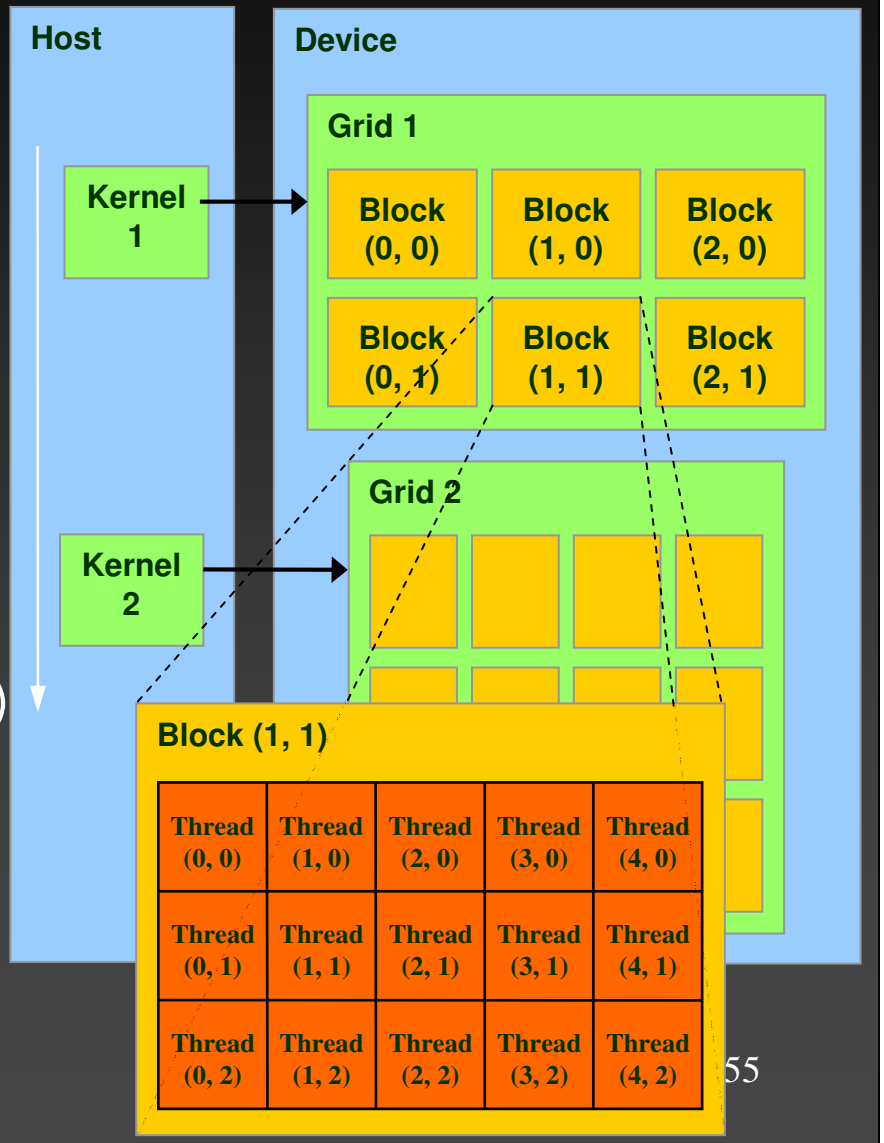
Rapports avec les architectures existantes

- Nvidia
  - G80, G92, GT200
- AMD / ATI
  - R600, R700

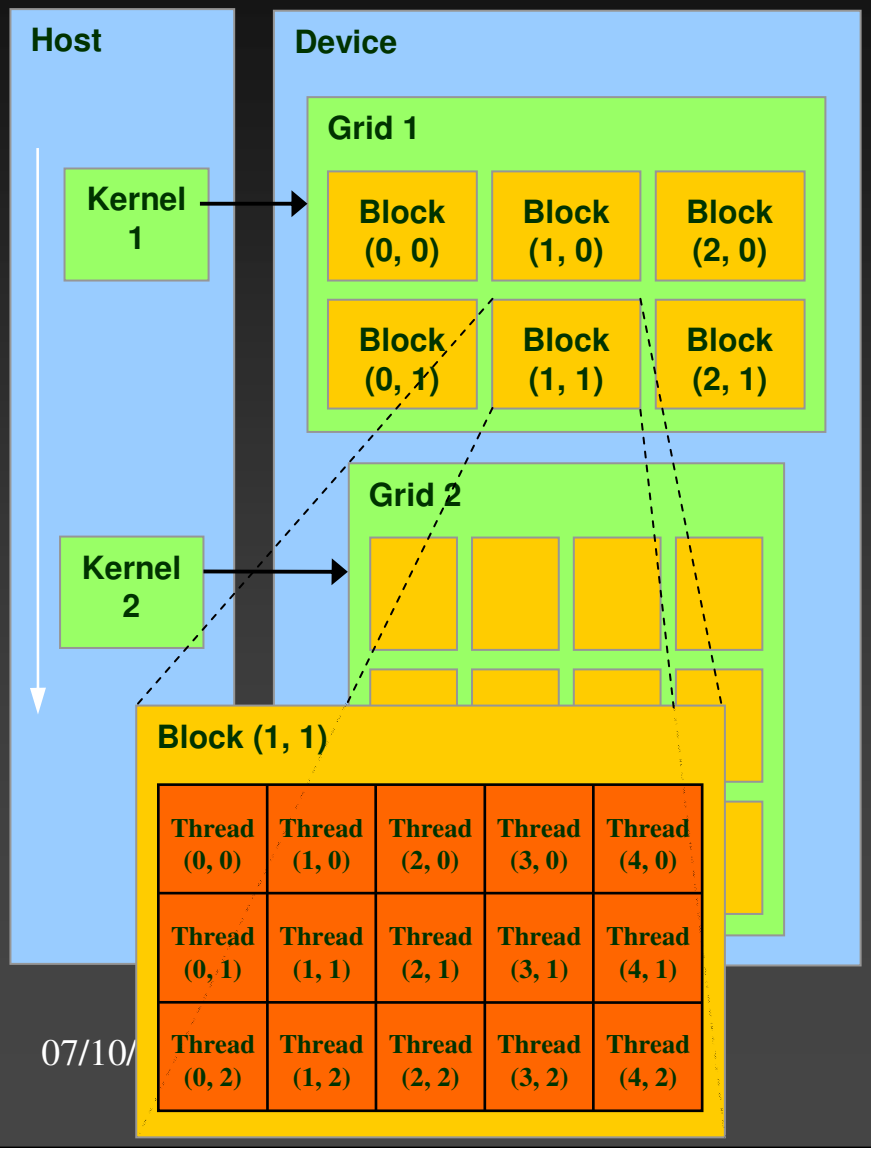
- Composants communs supplémentaires
  - Registres et mémoires de constantes
  - Mémoire et cache de textures

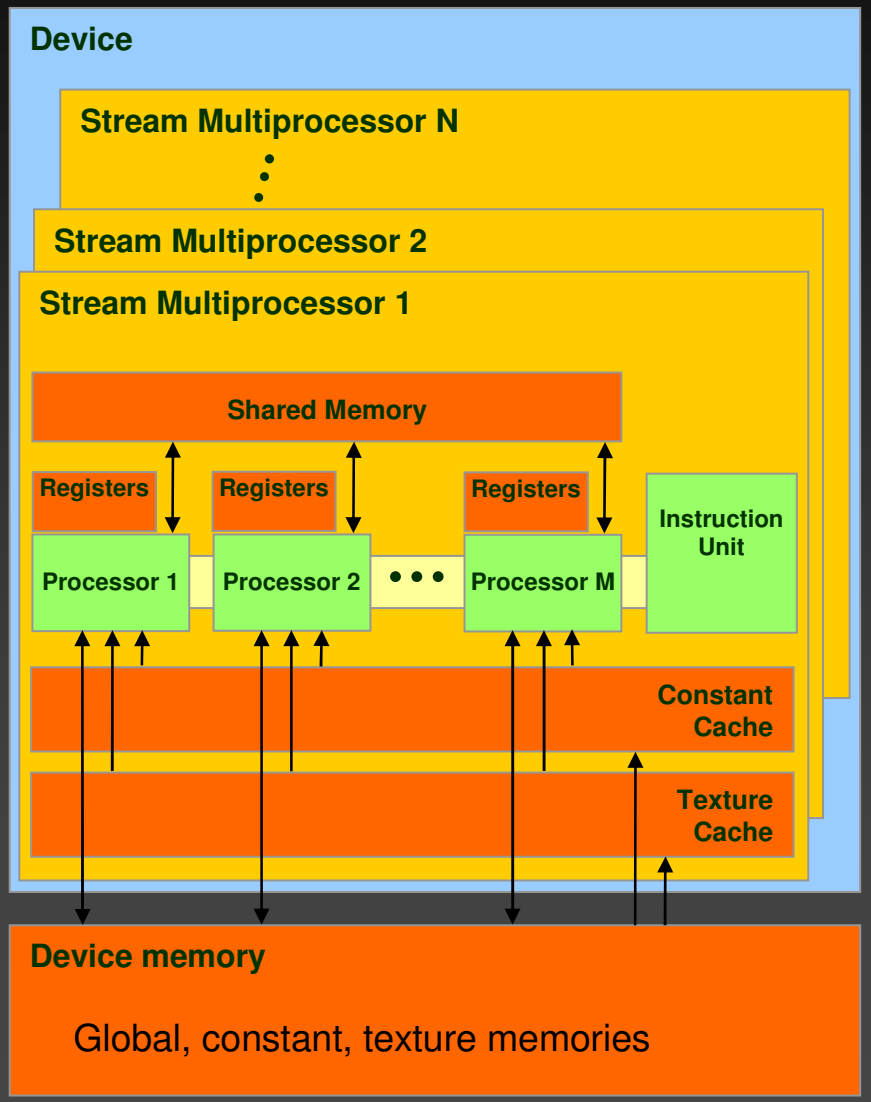
CUDA et mode vectoriel

- Vecteur logique = bloc
- Vecteur physique = warp
- Taille des VFU
  - 8 MAD (x 2 fréquence)
  - 2 SFU (x 2 fréquence)
- Multi-tâche
  - 256-512 reg. vec. à partager
  - 16ko de mem partagée
- Concept de processeur virtuel (SIMT)
- Accès mémoires regroupés (différents motifs acceptés)



Rapport avec CUDA



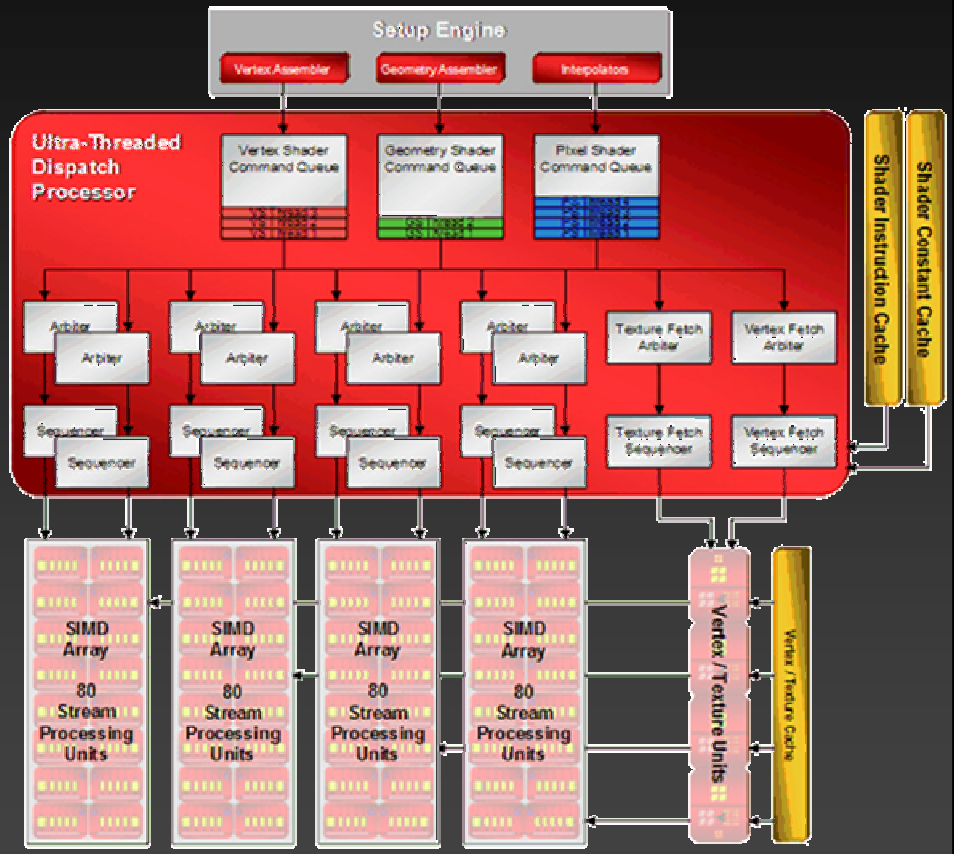


Quelques mots sur CUDA

- Découpage du travail fait par le développeur
  - Compilateur + simple
    (techniques de compilation éprouvées)
  - Contrôle de l'exécution des kernels
    (communication inter-intra kernel)
  - Contrôle sur le regroupement en vecteur physique
    (warp, gestion fine des aléas)
  - Gestion des registres divisée entre développeur et compilateur
    (effets indésirables comme la sur-utilisation)

Rapport avec le R600 et Brook+

- Vecteur logique = totalité du travail
- Vecteur physique = *wavefront* (taille du SIMD array * 4)
- Taille des VFU
  - 16 processeurs logiques VLIW à 4+1 voies (4 MAD, 1 SFU)
- 127 Registres vectoriels de 128 bits (VLIW)
  - Adressable en absolu, relatif à l'index d'une boucle, relatif à l'un des 4 registres d'adresse
- Registre de bypass géré par le compilateur
- Ring bus sur le R600 abandonné sur le R700 (techno complexe + conso)



ATI Stream Computing

- Découpage du travail laissé au compilateur/ processeur
  - Langage plus simple mais compilateur plus complexe (moins de calcul compliqué d'indice)
  - Compilateur prend en charge la totalité des registres (mise en place de techniques plus complexes que Nvidia, registre bypass, registres partagés)
  - Vectorisation explicite nécessaire (utilisation de float4 liée à l'architecture VLIW)
  - Pas de contrôle fin des aléas
  - Pas de contrôle de l'exécution des threads (mais connaissance de l'algo utilisé par le rasterizer)

Chapitre 5

Mesure de performance du modèle vectoriel

Étude de cas

- L'exécution d'une instruction sur chaque élément nécessite:

$$\frac{\text{(\# éléments x \# d'instruction)}}{\text{(\# d'éléments traités en // x fréquence)}}$$

  - Ce que font les constructeurs:
    - Augmentation du nombre d'éléments traités en //
      (+ de VFU / VMU, + de pipeline)
    - Augmentation de la fréquence

- Exemple : 2 M d'éléments et 1 instruction
  - Calcul :
    - GTX260 : (2M * 1) / (216 * 1242Mhz) = 0.0074 ms
    - HD4770 : (2M * 1) / (128*5 * 700Mhz) = 0.0045 ms
  - Chargement local :
    - GTX260 : (2M * 1) / (28 * 1242Mhz) = 0.057 ms
    - HD4770 : (2M * 1) / (16 * 700Mhz) = 0.178 ms

Mais ce qui compte beaucoup …

- Taille et nombre de vecteurs logiques
  - Dépend du nombre de registres nécessaire à l'exécution du kernel
- Taille et nombre de vecteurs physiques
  - Conditionne une partie granularité des aléas
- Taille des kernels
  - Overhead si trop petit ou trop grand
- Taille des bus et type de mémoire
- Ratio VMU / VFU et #inst arithmétique / #inst L / S
- Motif d'accès mémoire

- Critères classiques
  - Dépendance de données
  - Nb d'instructions

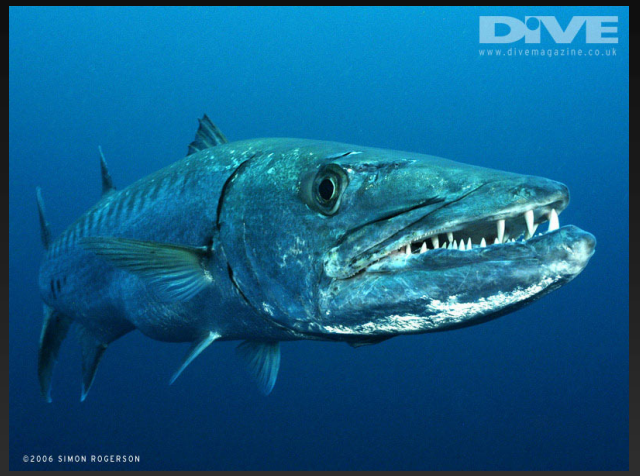
Rien de mieux qu'un test grandeur nature

- Comme avec les superscalaires il devient difficile de prédire les performances d'un code donné
- Debuggage et test en mode émulation (CPU)
  - Ne permet pas de tout observer (problème relatif à la performance)
- Solution : simulateur de processeur vectoriel
  - Les constructeurs en ont un mais il est secret
  - Émulateur dédié (… Barra)

Langages orientés flux

- Extensions C / C++
- Non commerciaux
  - Brook language (Stanford)
  - Sh library (University of Waterloo)
  - StreamIt (MIT)
  - OpenCL (Apple, Nvidia, ATI, …)
  - DirectX 11 (Microsoft)
- Commerciaux
  - TStreams
  - RapidMind
  - AccelerEyes
  - HMPP
- Fabriquants
  - Brook+ (AMD/ATI)
  - StreamC (Projet Imagine de Stanford)
  - Intel Ct (Intel - C for Throughput Computing)
  - CUDA (Nvidia)

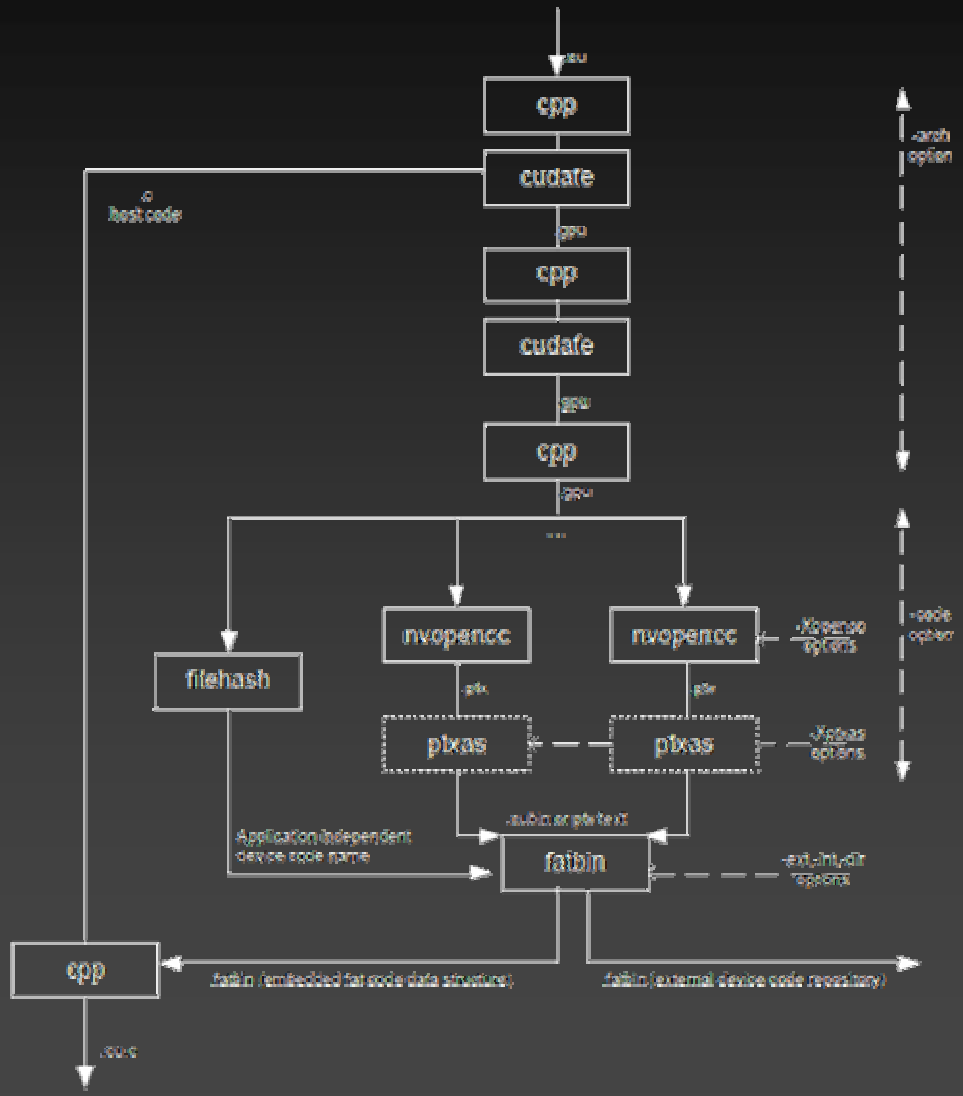
Barra, Simulateur Fonctionnel pour CUDA



- Simulation
  - Niveau cycle
    - Précis mais lent
  - Niveau transaction
    - Basée sur les communications
  - Niveau fonctionnel
    - Reproduit le comportement du processeur

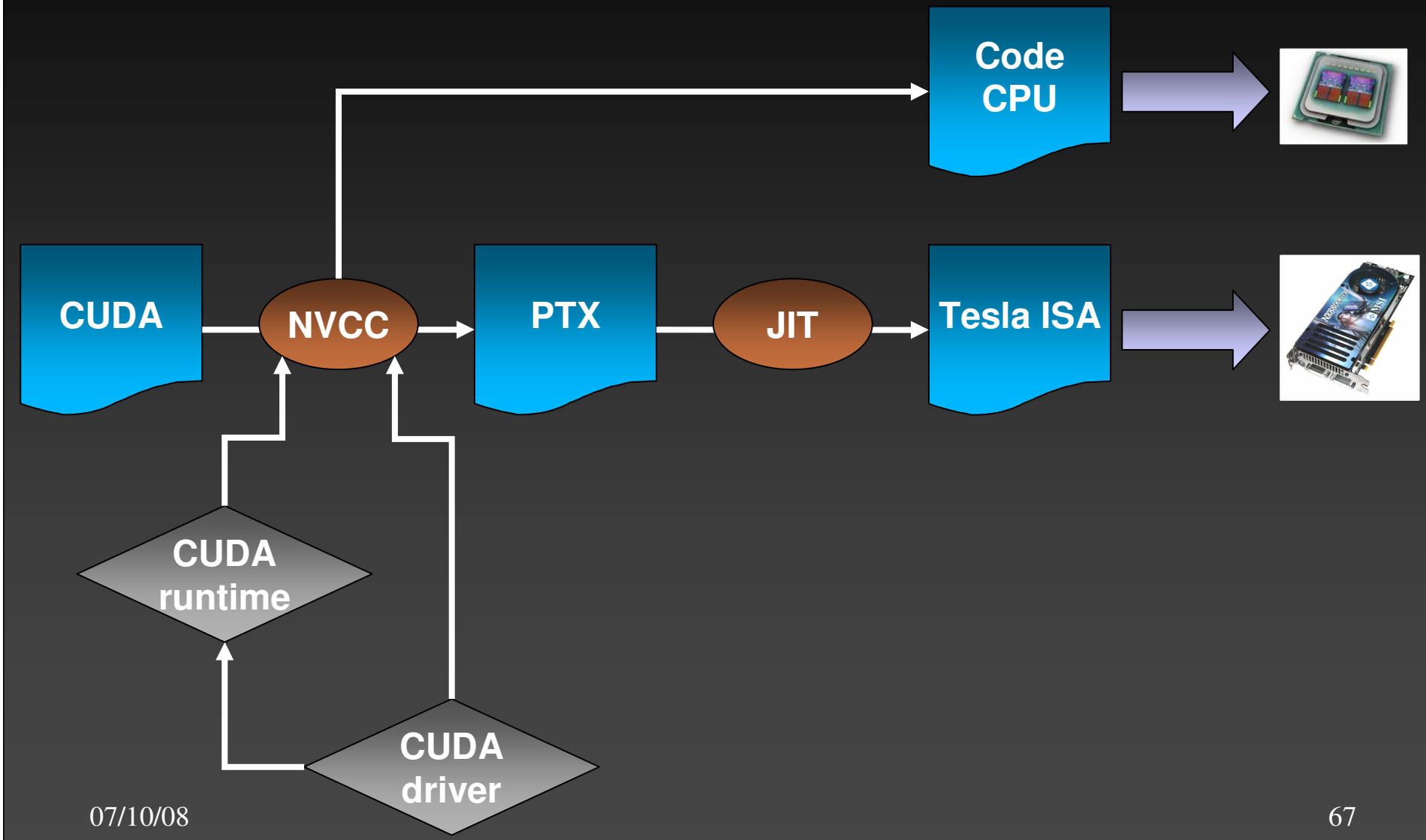
http://gpgpu.univ-perp.fr/index.php/Barra

Chaîne de compilation CUDA

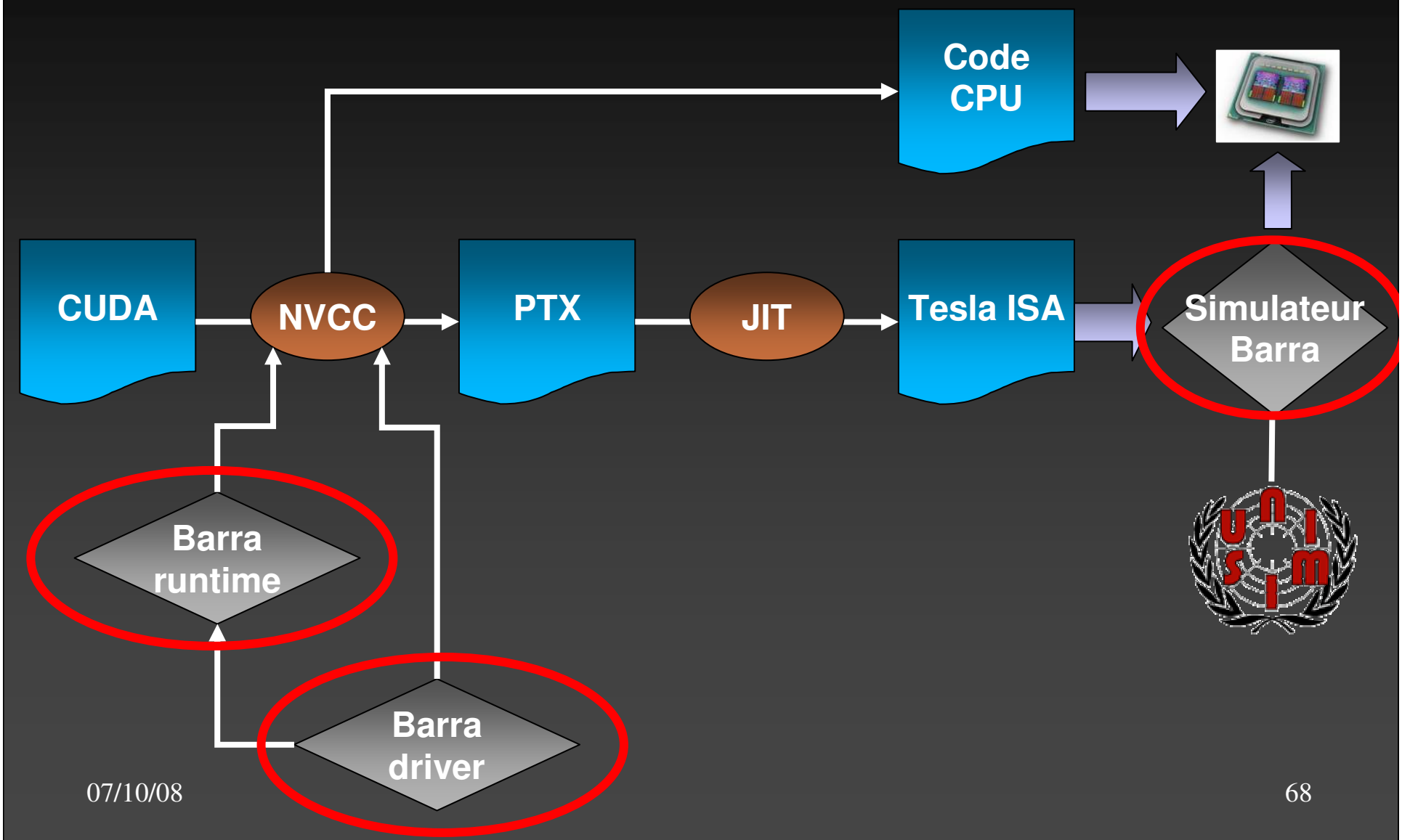


« The CUDA Compiler Driver NVCC », Nvidia tech. Doc.

Compilation CUDA



Compilation Barra



Simulation fonctionnelle

- Debbugage
  (dump des registres, mémoire partagée, …)
- Permet de tester son code en faisant varier des paramètres
  - # et tailles des registres
  - Fonctionnement des unités
    (utilisation de modèle de temps / conso)
- Exploration architecturale

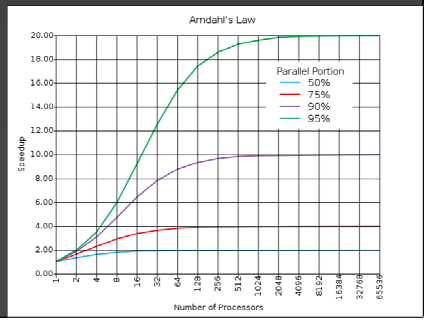
Conclusion

Avantages du vectoriel

- Facile d'avoir de très bonnes performances si
  - Les opérations sont indépendantes
  - On utilise les mêmes UF
  - Accès à des registres disjoints
  - Accès aux registres dans le même ordre que l'instruction précédente
  - Accès à des données contigues en mémoire ou avec des motifs connus
  - Exploite facilement une large bande passante
  - Cache facilement les latences d'accès à la mémoire et toutes les autres latences par ailleurs
- Passe à l'échelle
  - Plus on rajoute de ressources et plus on va vite
- Mode de programmation compact
  - Décrit N opérations en 1 instructions
- Plus prédictif
  - Meilleur pour le temps réel (moins de cache
- Technique de compilation connue

Attention, virage dangereux !

- Se concentrer sur les performances de crêtes et ignorer le surcoût de démarrage
- Augmentation des performances vectorielles sans regarder les performances du code scalaire (Amdahl veille sur les performances)
- Augmentation des performances de calcul sans améliorer l'accès à la mémoire !
  - MMX ?



Le futur… (sujets de recherche)

- Utilisation efficace des many-coeurs
  (quels critères, langages,… pour manipuler ces architectures)
- Gestion fine de la précision
  (garantir la fiabilité en minimisant l'utilisation des fp64, fp128)
- Algorithmes adaptatifs
  (gestion d'architectures diverses, SIMD, MIMD,…)
- Algorithmes résistants aux fautes
  (comment gérer l'apparition d'erreurs de calcul, pannes matérielles,…)

Pour les streamophiles

- Mark D. Hill, *Amdahl's Law in the Multicore Era*
  http://www.youtube.com/watch?v=KfgWmQpzD74&feature=channel
- Jack Dongarra, *An Overview of High Performance Computing and Challenges for the Future*
  http://www.youtube.com/watch?v=zTIKUxO9kf4&feature=related
- Dave Patterson, *Computer Architecture is Back: Parallel Computing Landscape*
  http://www.youtube.com/watch?v=On-k-E5HpcQ&feature=related
- William Dally, *Stream Computing*
  http://www.youtube.com/watch?v=8x7OqjUNbyo&feature=channel
- John Nickolls, *Scalable Parallel Programming with CUDA on Manycore GPUs*
  http://www.youtube.com/watch?v=nlGnKPpOpbE&feature=related