



# Programmation Parallèle et Distribuée

---

Patrick Carribault

David Dureau

Marc Pérache ([marc.perache@cea.fr](mailto:marc.perache@cea.fr))



# Supports

---

- [https://drive.google.com/drive/folders/1G8D95CCeyoLslLZUp4Uy8Vz\\_wyK5rBTG?usp=sharing](https://drive.google.com/drive/folders/1G8D95CCeyoLslLZUp4Uy8Vz_wyK5rBTG?usp=sharing)





# Contexte

---

- Evolution des architectures de processeurs ?
  - Augmentation de la fréquence
    - Ex : 1GHz → 1 milliard de changement d'horloge par seconde
  - Mais : limite physique
    - Consommation électrique
    - Dissipation de la chaleur
    - taille de gravure vs. taille d'un électron
- Solution :
  - Le parallélisme est la solution pour augmenter la puissance des processeurs
  - Plusieurs pistes...



# Contexte

---

- Parallélisme
  - Déjà existant au sein d'un processeur (pipeline, traitement de plusieurs instructions, exécution Out-Of-Order, ...)
  - Multiplication des unités de traitements
    - Augmentation du nombre de coeurs
    - Duplication des unités vectorielles
- De nombreux domaines utilisent des ordinateurs "massivement parallèle" (calcul haute performance) :
  - simulation numérique (Industries aéronautique, automobile, nucléaire, Météorologie...) ;
  - infographie : films d'animation ;
  - traitement d'image (Photoshop) ;



# But de ce cours

---

- Comprendre le parallélisme
  - Description d'une architecture de processeur/nœud de calcul
  - Découverte des types de parallélisme
- Apprendre à exploiter le parallélisme d'un code
  - Trouver le parallélisme
  - Connaître les modèles de programmation
- Résumé
  - "Comprendre le parallélisme et savoir programmer des applications parallèles est un atout majeur"



# Déroulement du module

---

- Prérequis
  - Système d'exploitation : Linux
  - Langage de programmation : C
    - Maîtrise arithmétique pointeur demandée
- Travail en salle machine
  - Programmation en parallèle
- Evaluation des connaissances
  - Partiels & TPs



# Plan

---

- Introduction
  - Architectures et programmation
- Programmation mémoire distribuée
  - Modèle MPI (*Message-Passing Interface*)
- Programmation mémoire partagée
  - Modèle *thread*
  - Modèle OpenMP
- Vers des modèles hybrides et hétérogènes



# Plan du cours 1

---

- Architecture des machines parallèles
  - Système à mémoire partagée
  - Système à mémoire distribuée
  - Supercalculateurs
- Introduction à la programmation parallèle
  - Notions et définitions
  - Types de parallélisme
  - Modèles de programmation



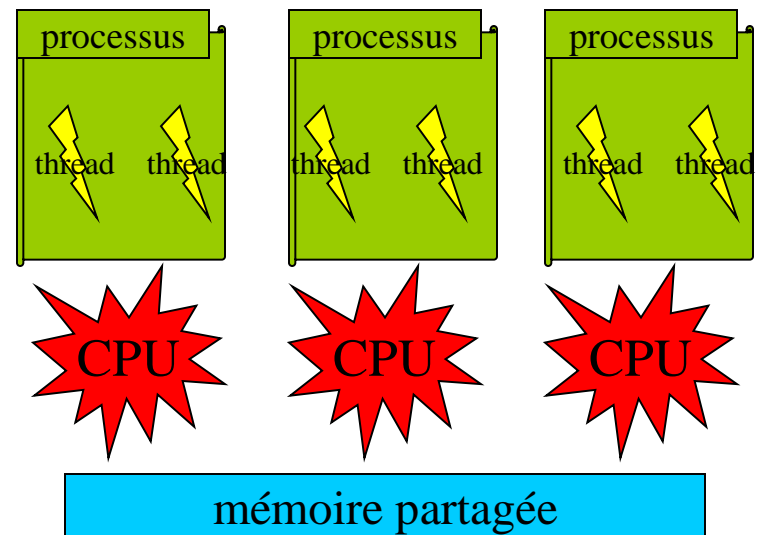
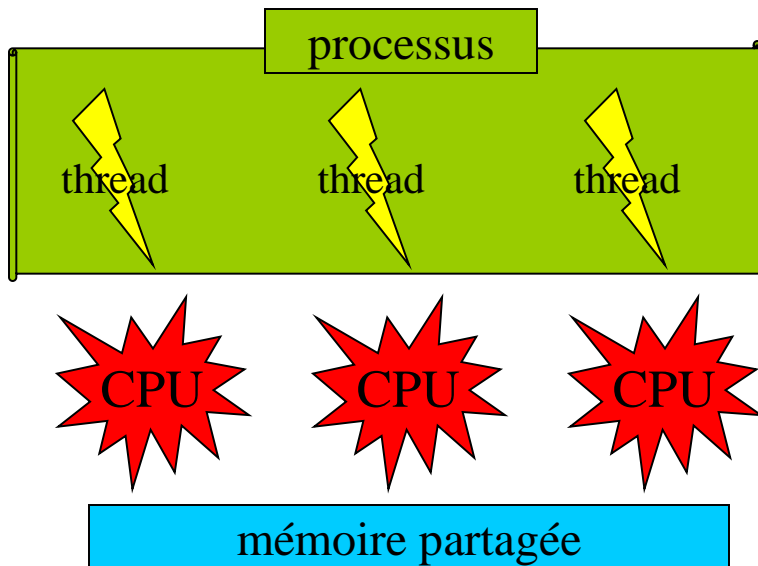
# Architecture des machines parallèles



---

# Systeme à memoire partagee

- Systeme à memoire partagee
  - Systeme mettant en jeu plusieurs ressources de calcul qui ont de la memoire partagee (physiquement ou de maniere logicielle)
- Noeud
  - Plus grand ensemble de processeurs partageant materiellement de la memoire. On parle de noeud SMP ou de noeud NUMA





# Systeme à mémoire partagée

---

- Principaux types de systèmes à mémoire partagée :
  - SMP & NUMA
- SMP (*Symmetrical Multi-Processing*)
  - Machine constituée de plusieurs processeurs identiques connectés à une unique mémoire physique (de l'ordre de 2 à 4 processeurs);
- NUMA (*Non-Uniform Memory Access*)
  - Machine constituée de plusieurs processeurs connectés à plusieurs mémoires distinctes (on parle de bancs mémoires) reliées entre elles par des mécanismes matériels (jusqu'à 2048 processeurs)
- Remarque : d'un point de vue OS, il existe une mémoire unique dont la taille est la somme des mémoires physiques distinctes.

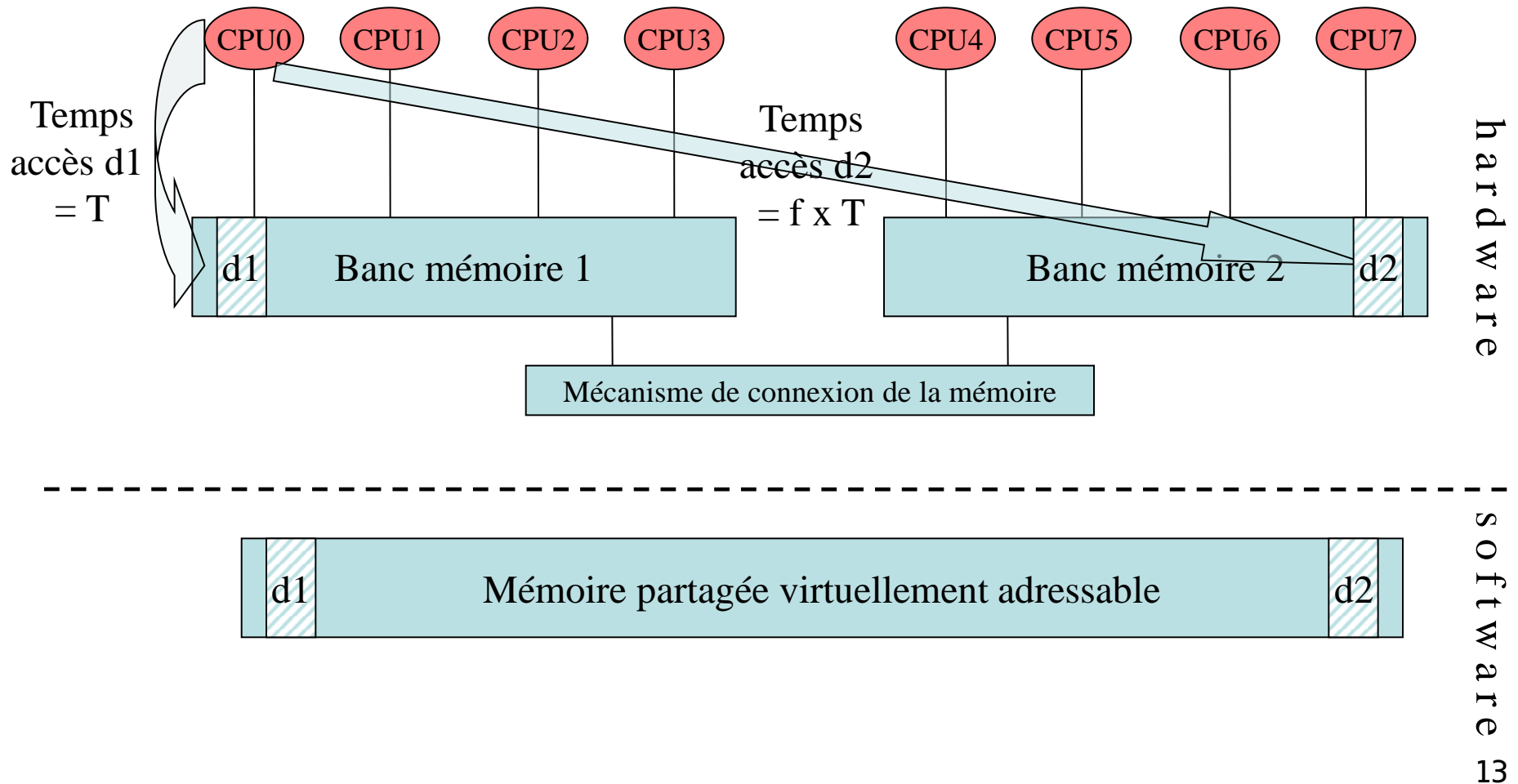


# Hiérarchie mémoire

---

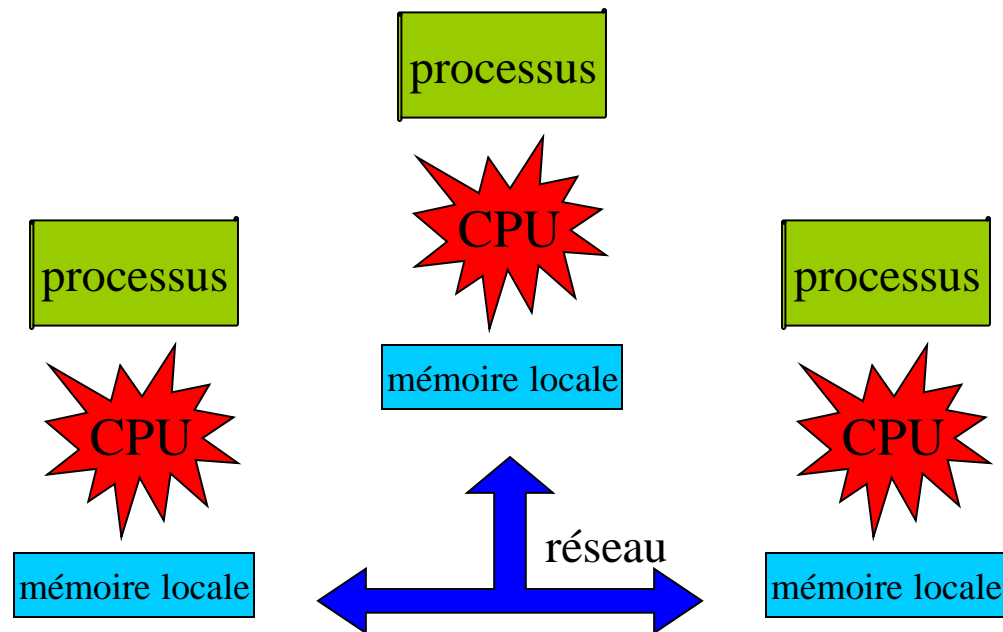
- Différence majeure entre une machine SMP et une machine NUMA : hiérarchie de la mémoire
- SMP : tous les processeurs accèdent à n'importe quelle case de la mémoire en un temps uniforme (i.e., de façon symétrique);
- NUMA : le temps d'accès à la mémoire dépend du placement des données dans celle-ci :
  - rapide si dans le banc mémoire local au processeur
  - lent (d'un facteur appelé *facteur NUMA*) si dans un autre banc mémoire
  - Remarque : plusieurs niveaux ( $> 2$ ) de localité des données existent en fonction des machines

# Hiérarchie mémoire



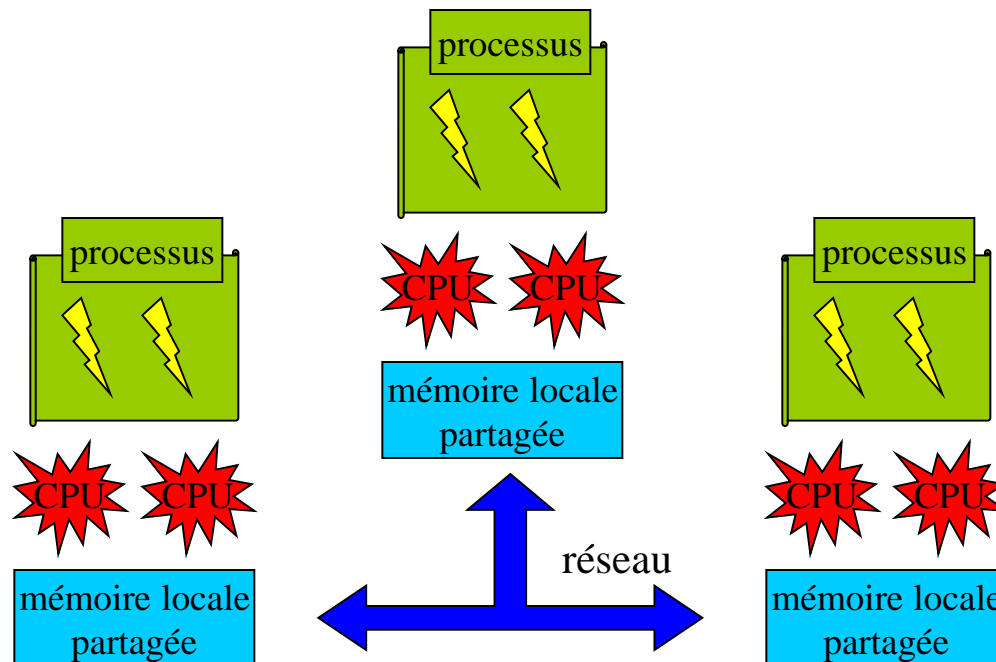
# Systeme à mémoire distribuée

- Systeme à mémoire distribuée
  - Ressources de calcul qui n'ont pas de mémoire partagée, que ce soit de manière physique ou logicielle

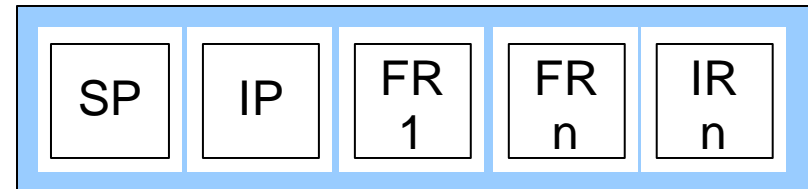
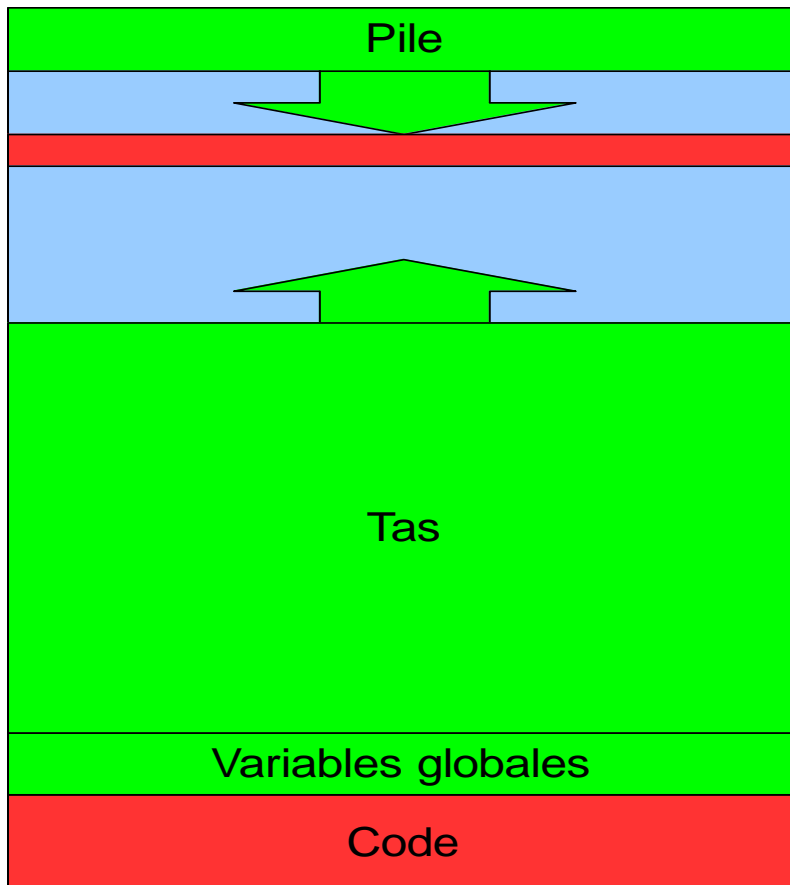


# Systeme hybrides

- Hybrides : mémoire partagée/distribuée
- Grappe ou *Cluster*
  - Ensemble de nœuds interconnectés par un réseau



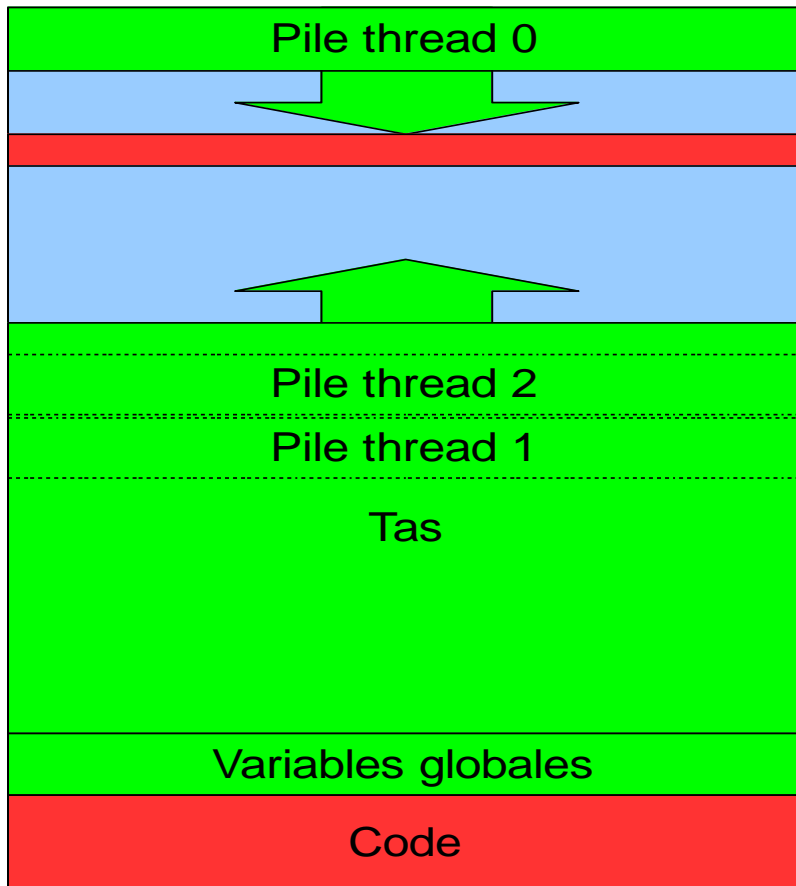
# Processus



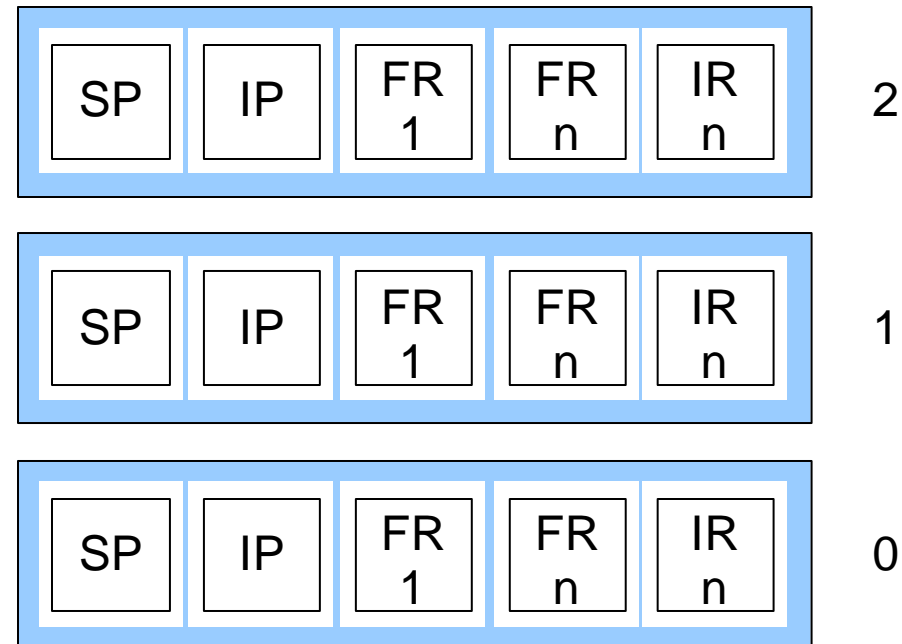
Structures



# Processus multithread



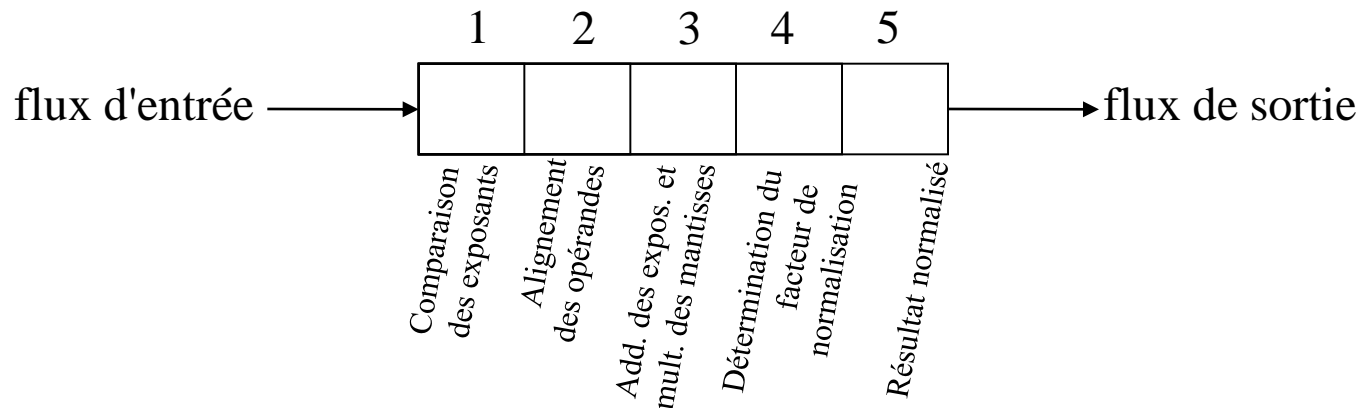
Mémoire

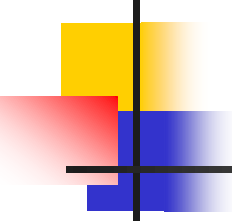


Structures

# Evolution des supercalculateurs

- Première révolution : les machines vectorielles
- 1965 - ILLIAC IV (Burrough) : échec commercial
- 1976 - Cray 1 : succès commercial
  - Un calculateur vectoriel dispose d'unités fonctionnelles segmentées. Une unité fonctionnelle exécute une instruction complexe telle qu'une addition ou une multiplication entière ou flottante. Une opération arithmétique de ce genre nécessite plusieurs instructions élémentaires





# Evolution des supercalculateurs

---

- Deuxième révolution : apparition du parallélisme
- Fin des années 80 – début des années 90 : recul du vectoriel + apparition des microprocesseurs
  - expérience parallélisme massif : Connection Machine (65536 processeurs)
  - apparition des systèmes partagés avec un nombre raisonnable de processeurs
  - apparition des systèmes distribués avec un grand nombre de processeurs, mais sans mémoire partagée
- Début des années 2000 : systèmes hybrides à mémoire distribuée/partagée
  - clusters de SMP puis de NUMA



# Evolution des supercalculateurs : tendance actuelle

---

- Apparition des processeurs multicoeurs
  - CPU généralistes à plusieurs cœurs de calcul
  
- Evolution sur la taille des noeuds NUMA :
  - Nombre de cœurs croissant : exemple pour le CEA/DAM
    - 16 coeurs sur un nœud standard Tera10,
    - 32/128 coeurs sur un nœud Tera100
  
  - Mémoire par coeur diminuant :
    - 3 Go/coeur pour Tera10 (48 Go pour le noeud entier),
    - 2 Go/coeur pour Tera100 (64 Go pour le noeud entier)



# Pourquoi les processeurs multicores sont ils apparus ?

---

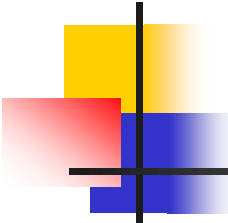
- Augmentation la fréquence des processeurs "classiques":
  - aux limites de la loi de Moore
    - Temps d'interconnexion entre transistors ne diminuent plus avec la finesse de gravure (trop petite)
    - Consommation électrique et la dissipation thermique explosent (fonction du carré du nb de transistors) => aujourd'hui, des processeurs consomment jusqu'à 100 W et nécessitent des systèmes de refroidissement complexes
    - Fréquence de la mémoire n'augmente pas dans les mêmes proportions que la fréquence du processeur => l'accès à la mémoire devient pénalisant (beaucoup de cycles perdus à accéder à la mémoire). Aujourd'hui, plus de mémoire cache, mais pas suffisant
  - aux limites de l'ILP (Instruction Level Parallelism)
    - extraire du parallélisme du flot d'exécution + réordonnancement des instructions => rapidement limité par la complexité des algorithmes ou par la profondeur des pipelines
- => Il faut un saut technologique



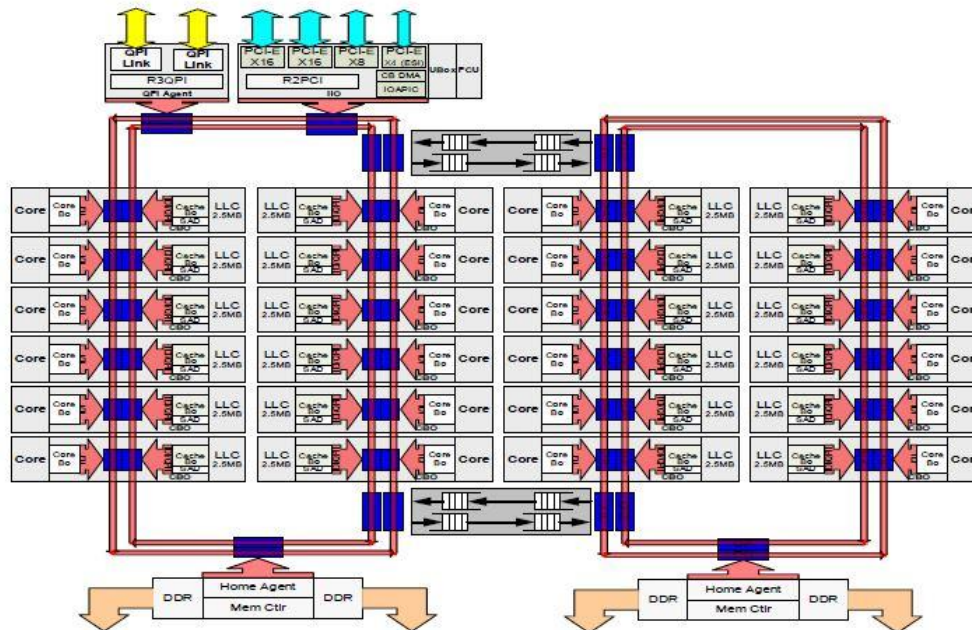
# Processeurs multicores

---

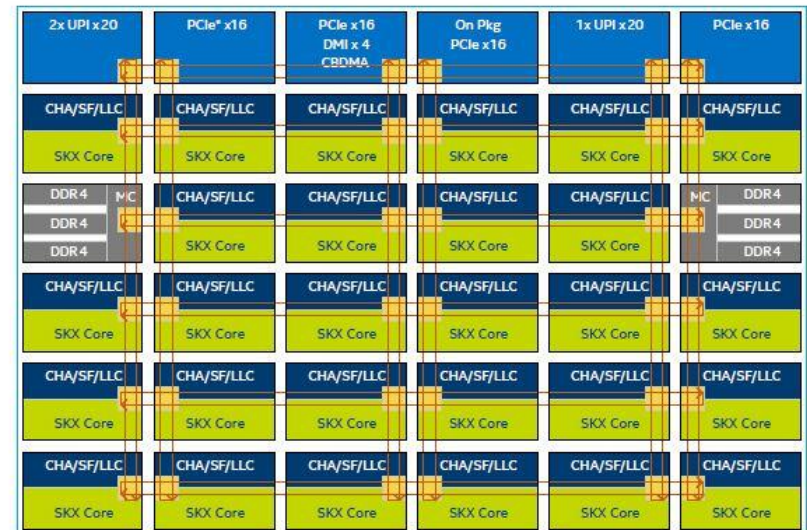
- Processeur multicores
  - Parallélisme intensif au niveau du processeur
  - concentration sur une même puce de plusieurs processeurs "classiques" (appelés désormais cores ou cœurs) + mécanisme matériel pour gérer plusieurs caches, des accès concurrents à la mémoire, etc..
  - reproduction d'un nœud SMP au niveau du processeur
- Gains apportés
  - en théorie, un processeur n-cores est n fois plus puissant qu'un processeur monocore à la même fréquence
  - en pratique, selon la conception de l'architecture (cache partagée ou non, nb de switch pour accéder à la mémoire, existence communications directes entre cœurs) les performances peuvent être bien en deçà



## Broadwell EX 24-core die



## Skylake-SP 28-core die



CHA – Caching and Home Agent ; SF– Snoop Filter; LLC – Last Level Cache;  
SKX Core– Skylake Server Core; UPI – Intel® UltraPath Interconnect

## MESH IMPROVES SCALABILITY WITH HIGHER BANDWIDTH AND REDUCED LATENCIES

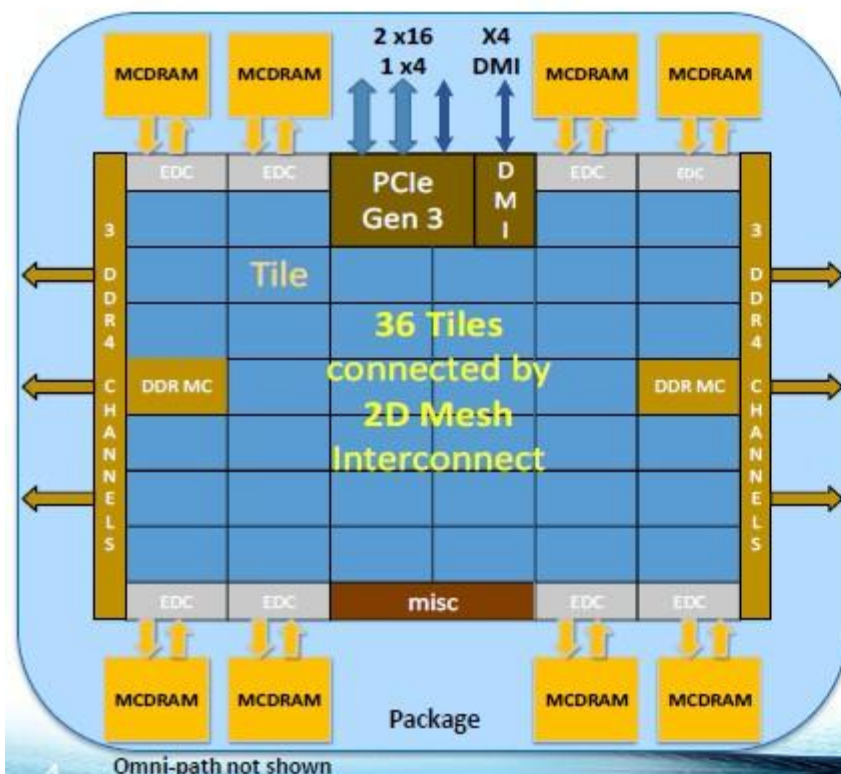


# Intel KNL

## Knights Landing Overview

### TILE

2 VPU	CHA	2 VPU
Core	1MB L2	Core



Omni-path not shown

**Chip: 36 Tiles** interconnected by 2D Mesh

**Tile: 2 Cores + 2 VPU/core + 1 MB L2**

**Memory: MCDRAM: 16 GB on-package; High BW**

**DDR4: 6 channels @ 2400 up to 384GB**

**IO: 36 lanes PCIe Gen3. 4 lanes of DMI for chipset**

**Node: 1-Socket only**

**Fabric: Omni-Path on-package (not shown)**

**Vector Peak Perf: 3+TF DP and 6+TF SP Flops**

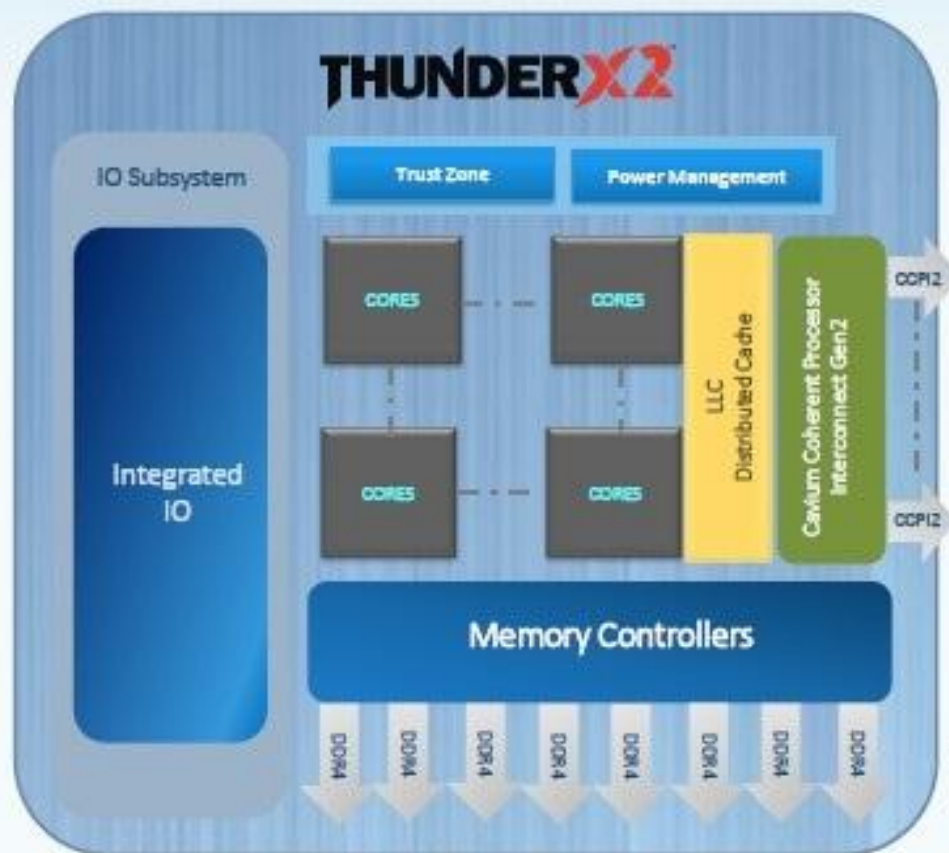
**Scalar Perf: ~3x over Knights Corner**

**Streams Triad (GB/s): MCDRAM : 400+; DDR: 90+**

Source Intel: All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. KNL data are preliminary based on current expectations and are subject to change without notice. 1Binary Compatible with Intel Xeon processors using Haswell architecture (Sagebrush PEK). Bandwidth numbers are based on STREAM-like memory access pattern where MCDRAM used as local memory. Results have been estimated based on internal Intel analysis and are not intended for commercial purposes only. Any difference in system results or software stack may affect memory access performance.



# ARM



- 24/28/32 Custom ARMv8 cores
- Fully Out-Of-Order (OOO) Execution
- 1S and 2S Configuration
- Up to 8 DDR4 Memory Controllers
- Up to 16 DIMMs per Socket
- Server Class RAS features
- Server class virtualization
- Integrated IOs
- Extensive Power Management

2<sup>nd</sup> gen Arm server SoC

Delivers **2-3X** higher performance



# Comment exploiter les processeurs multicores ?

---

- Constat

- Une application séquentielle n'exploite pas naturellement la pleine puissance des processeurs multicores (car elle s'exécute sur un seul cœur)

- Bilan

- les architectures même des processeurs allant vers plus de parallélisme
- le gain en performance n'est plus "gratuit"
- les applications séquentielles doivent être portées (on parle de paralléliser)
- il faut repenser les applications, penser parallélisme

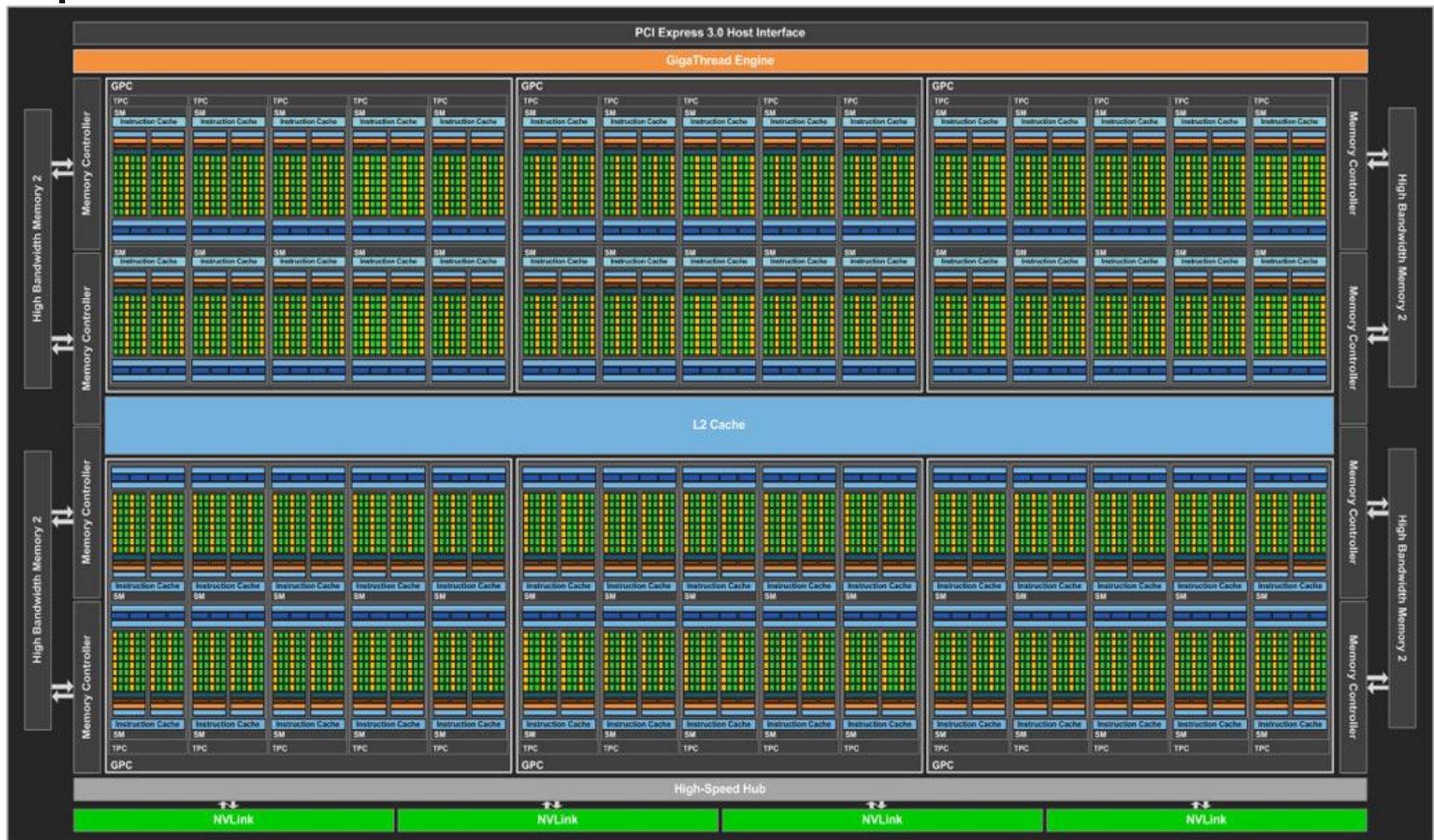


# Evolution des supercalculateurs : tendance actuelle

---

- Apparition d'accélérateurs GPGPU (General Purpose computing on Graphics Processing Units) :
  - Meilleur rapport consommation électrique/puissance calcul
  - Exemple : Tesla, Fermi (Nvidia), ATI Stream (AMD-ATI)
- Architecture :
  - Mémoire fortement hiérarchisée
  - « many-cores » : plusieurs centaines de coeurs SIMD
- Comment programmer ces cartes ? Pas facile
  - Programmation vectorielle adaptée (grâce aux instructions SIMD)
  - Chaque constructeur fournit son propre langage : CUDA pour Nvidia (proche du C), CAL/IL pour AMD-ATI
  - Naissance d'un langage unifié : OpenCL 1.0
- Pilotage depuis une machine (généraliste) hôte :
  - transfert de données (mémoire hôte => mém. GPGPU, mém. GPGPU => mém. Hôte) plutôt lent pour l'instant (connexion PCI)
  - Lancement depuis l'hôte des noyaux de calcul (kernel) sur la carte

# Architecture NVidia



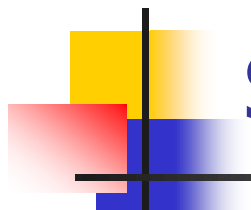


# Evolution des supercalculateurs : tendance actuelle

---

- Les supercalculateurs deviennent hybrides, hétérogènes
  - Des noeuds équipés de CPU multicoeurs généralistes
  - Des noeuds équipés de CPU + cartes accélératrices GPGPU
- Comment une application peut utiliser simultanément et efficacement des CPU généralistes et des GPGPU ?
  - Vrai défi, pas de réponse évidente, sujet ouvert ...
- => Troisième révolution en route : le multicoeurs et l'hybridation

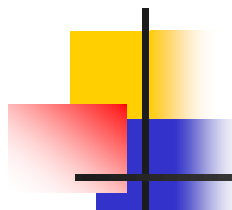
# Exemple d'architecture de supercalculateur



Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	<a href="#">Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11</a> , HPE DOE/SC/Oak Ridge National Laboratory, United States	8,699,904	1,194.00	1,679.82	22,703
2	<a href="#">Aurora - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11</a> , Intel DOE/SC/Argonne National Laboratory, United States	4,742,808	585.34	1,059.33	24,687
3	<a href="#">Eagle - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR</a> , Microsoft Microsoft Azure, United States	1,123,200	561.20	846.84	
4	<a href="#">Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D</a> , Fujitsu RIKEN Center for Computational Science, Japan	7,630,848	442.01	537.21	29,899
5	<a href="#">LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11</a> , HPE EuroHPC/CSC, Finland	2,752,704	379.70	531.51	7,107
6	<a href="#">Leonardo - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband</a> , EVIDEN EuroHPC/CINECA, Italy	1,824,768	238.70	304.47	7,404
7	<a href="#">Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband</a> , IBM DOE/SC/Oak Ridge National Laboratory, United States	2,414,592	148.60	200.79	10,096
8	<a href="#">MareNostrum 5 ACC - BullSequana XH3000, Xeon Platinum 8460Y+ 40C 2.3GHz, NVIDIA H100 64GB, Infiniband NDR200</a> , EVIDEN EuroHPC/BSC, Spain	680,960	138.20	265.57	2,560
9	<a href="#">Eos NVIDIA DGX SuperPOD - NVIDIA DGX H100, Xeon Platinum 8480C 56C 3.8GHz, NVIDIA H100, Infiniband NDR400</a> , Nvidia NVIDIA Corporation, United States	485,888	121.40	188.65	
10	<a href="#">Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband</a> , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL, United States	1,572,480	94.64	125.71	7,438
11	<a href="#">Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway</a> , NRCPC National Supercomputing Center in Wuxi, China	10,649,600	93.01	125.44	15,371
12	<a href="#">Perlmutter - HPE Cray EX 235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-11</a> , HPE DOE/SC/LBNL/NERSC, United States	888,832	79.23	113.00	2,945



# Exemple d'architecture de supercalculateur



Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
13	<a href="#">Selene - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband</a> , Nvidia <a href="#">NVIDIA Corporation</a> , United States	555,520	63.46	79.22	2,646
14	<a href="#">Tianhe-2A - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000</a> , NUDT <a href="#">National Super Computer Center in Guangzhou</a> , China	4,981,760	61.44	100.68	18,482
15	<a href="#">Explorer-WUS3 - ND96_amsr_MI200_v4, AMD EPYC 7V12 48C 2.45GHz, AMD Instinct MI250X, Infiniband HDR</a> , Microsoft Azure <a href="#">West US3</a> , United States	445,440	53.96	86.99	
16	<a href="#">ISEG - Gigabyte G593-SD0, Xeon Platinum 8468 48C 2.1GHz, NVIDIA H100 SXM5 80 GB, Infiniband NDR400</a> , Nebius AI <a href="#">Nebius</a> , Netherlands	218,880	46.54	86.79	1,320
17	<a href="#">Adastra - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11</a> , HPE <a href="#">Grand Equipement National de Calcul Intensif - Centre Informatique National de l'Enseignement Suprieur (GENCI-CINES)</a> , France	319,072	46.10	61.61	921
18	<a href="#">JUWELS Booster Module - Bull Sequana XH2000 , AMD EPYC 7402 24C 2.8GHz, NVIDIA A100, Mellanox HDR InfiniBand/ParTec ParaStation ClusterSuite</a> , EVIDEN <a href="#">Forschungszentrum Juelich (FZJ)</a> , Germany	449,280	44.12	70.98	1,764
19	<a href="#">MareNostrum 5 GPP - ThinkSystem SD650 v3, Xeon Platinum 03H-LC 56C 1.7GHz, Infiniband NDR200</a> , Lenovo <a href="#">EuroHPC/BSC</a> , Spain	725,760	40.10	46.37	5,753
20	<a href="#">Shaheen III - CPU - HPE Cray EX, AMD EPYC 9654 96C 2.4GHz, Slingshot-11</a> , HPE <a href="#">King Abdullah University of Science and Technology</a> , Saudi Arabia	877,824	35.66	39.61	5,301
21	<a href="#">HPC5 - PowerEdge C4140, Xeon Gold 6252 24C 2.1GHz, NVIDIA Tesla V100, Mellanox HDR Infiniband</a> , DELL EMC <a href="#">Eni S.p.A.</a> , Italy	669,760	35.45	51.72	2,252
22	<a href="#">Sejong - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Infiniband HDR</a> , Nvidia <a href="#">NAVER Corp</a> , South Korea	277,760	32.97	40.77	
23	<a href="#">Voyager-EUS2 - ND96amsr_A100_v4, AMD EPYC 7V12 48C 2.45GHz, NVIDIA A100 80GB, Mellanox HDR Infiniband</a> , Microsoft Azure <a href="#">Microsoft Azure</a> , United States	253,440	30.05	39.53	

# The Sierra system that will replace Sequoia features a GPU-accelerated architecture



## Compute Node

2 IBM POWER9 CPUs  
4 NVIDIA Volta GPUs  
NVMe-compatible PCIe 1.6 TB SSD  
256 GiB DDR4  
16 GiB Globally addressable HBM2  
associated with each GPU  
Coherent Shared Memory

## Compute Rack

Standard 19"  
Warm water cooling

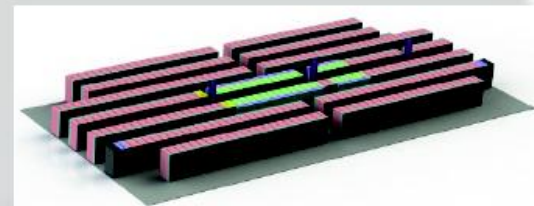
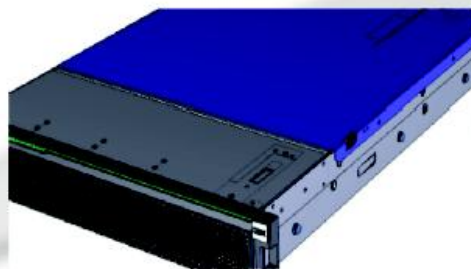
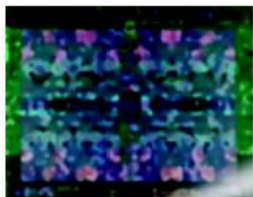
## Compute System

4320 nodes  
1.29 PB Memory  
240 Compute Racks  
125 PFLOPS  
~12 MW

## Components

### IBM POWER9

- Gen2 NVLink



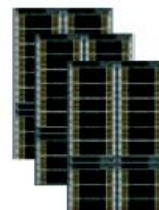
### NVIDIA Volta

- 7 TFlop/s
- HBM2
- Gen2 NVLink



### Mellanox Interconnect

Single Plane EDR InfiniBand  
2 to 1 Tapered Fat Tree



### Spectrum Scale File System

154 PB usable storage  
1.54 TB/s R/W bandwidth



# Sierra system architecture details



	Sierra	uSierra	rzSierra
Nodes	4,320	684	54
POWER9 processors per node	2	2	2
GV100 (Volta) GPUs per node	4	4	4
Node Peak (TFLOP/s)	29.1	29.1	29.1
System Peak (PFLOP/s)	125	19.9	1.57
Node Memory (GiB)	320(256+64)	320(256+64)	320(256+64)
System Memory (PiB)	1.29	0.209	0.017
Interconnect	2x IB EDR	2x IB EDR	2x IB EDR
Off-Node Aggregate b/w (GB/s)	45.5	45.5	45.5
Compute racks	240	38	3
Network and Infrastructure racks	13	4	0.5
Storage Racks	24	4	0.5
Total racks	277	46	4
Peak Power (MW)	~12	~1.8	~0.14



# EXA HF

Site: CEA-HF - BULLSEQUANA XH2000, AMD EPYC 7763 64C 2.45GHZ, ATOS BXI V2

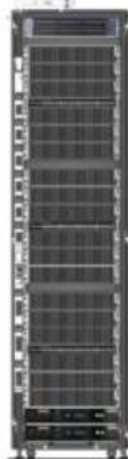
Site:	<a href="#">Commissariat a l'Energie Atomique (CEA)</a>
System URL:	<a href="http://www.cea.fr/">http://www.cea.fr/</a>
Manufacturer:	Atos
Cores:	810,240
Memory:	1,658,880 GB
Processor:	AMD EPYC 7763 64C 2.45GHz
Interconnect:	Atos BXI V2
Performance	
Linpack Performance (Rmax)	23,237.6 TFlop/s
Theoretical Peak (Rpeak)	31,761.4 TFlop/s
Nmax	12,153,600
HPCG [TFlop/s]	340.84
Power Consumption	
Power:	4,959.47 kW (Submitted)

# Astra – the First Petscale Arm based Supercomputer

**HPE Apollo 70 Chassis: 4 nodes**



**HPE Apollo 70 Rack**



18 chassis/rack

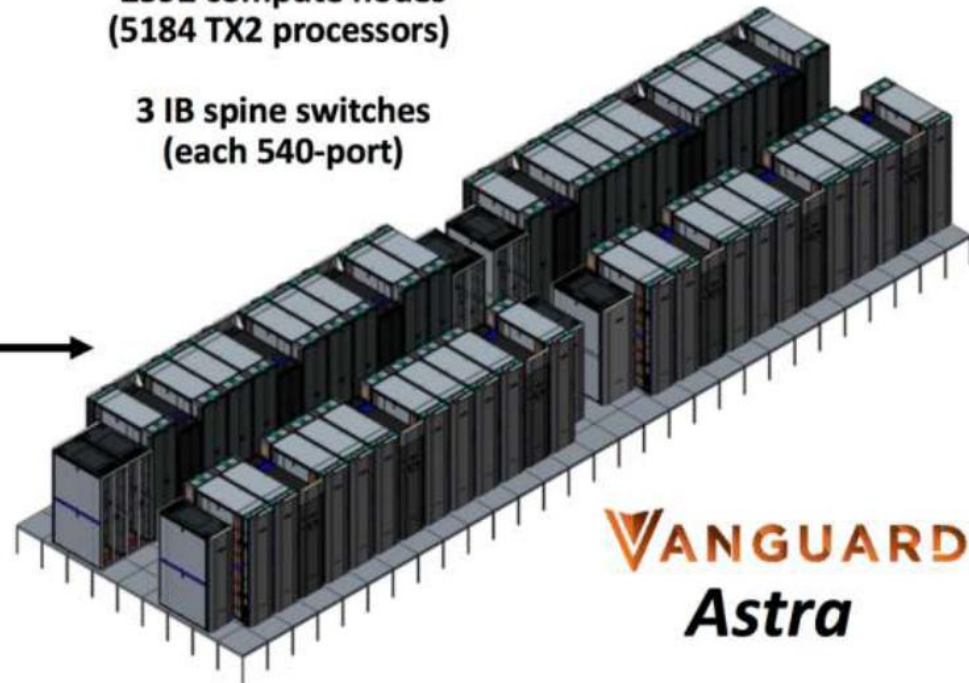
72 nodes/rack

3 IB switches/rack  
(one 36-port switch  
per 6 chassis)

36 compute racks  
(9 scalable units, each 4 racks)

2592 compute nodes  
(5184 TX2 processors)

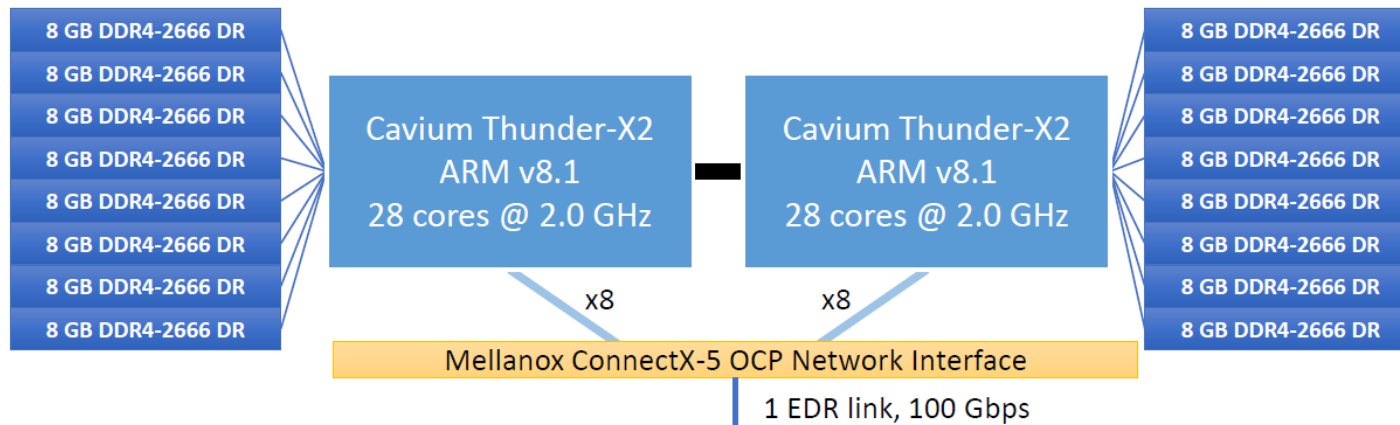
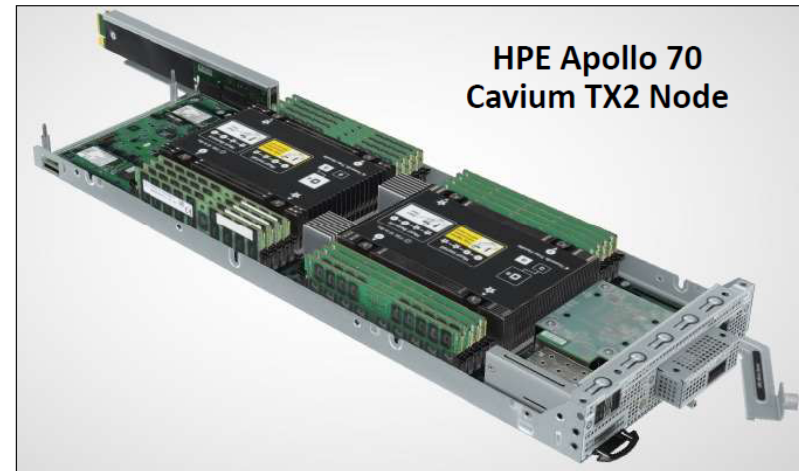
3 IB spine switches  
(each 540-port)



**VANGUARD**  
*Astra*

# Astra Architecture

- **2,592** HPE Apollo 70 compute nodes
  - Cavium Thunder-X2 **Arm** SoC, 28 core, 2.0 GHz
  - 5,184 CPUs, 145,152 cores, 2.3 PFLOPs system peak
  - 128GB DDR Memory per node (**8 memory channels per socket**)
  - Aggregate capacity: 332 TB, Aggregate Bandwidth: 885 TB/s
- Mellanox IB EDR, ConnectX-5
- HPE Apollo 4520 All-flash storage, Lustre parallel file-system
  - Capacity: 403 TB (usable)
  - Bandwidth 244 GB/s





# Résumé

---

- Architectures parallèles caractérisées par la topologie mémoire
  - Partagée, distribuée, partagée/distribuée;
- Aujourd'hui, le parallélisme est présent au sein même du processeur
  - Superscalaires
  - Multicoeurs
- Lors de l'écriture d'une application, il faut désormais penser parallélisme



# Introduction à la programmation parallèle

---



# Définitions

---

- Tâche
  - Travail à faire
- Thread (ou flot d'exécution)
  - Implémentation d'une tâche : suite logique séquentielle d'actions résultat de l'exécution d'un programme
- Processus
  - Instance d'un programme. Un processus est constitué d'un ou plusieurs threads qui partagent un espace d'adressage commun. Si un processus comporte plusieurs threads, il est dit multithread
- Calcul parallèle
  - Le calcul parallèle consiste en le découpage d'un programme en plusieurs tâches qui peuvent être exécutées en même temps dans le but d'améliorer le temps global d'exécution du programme



# Qu'est-ce que le parallélisme ?

---

- Vieille idée pour résoudre plus vite un problème long et coûteux en temps calcul;
- Une solution : utiliser plusieurs unités de traitement (i.e. processeurs);
- Difficulté : organisation des tâches parallèles (algorithmique parallèle) :
  - résoudre correctement le problème initial : relation de dépendances entre tâches;
  - les unités de traitement doivent avoir constamment du travail (utile) à effectuer : distribution et équilibrage (dynamique) de la charge;





# Programmation séquentielle et parallèle

---

- Programmation séquentielle :
  - Suite ORDONNÉE d'instructions à exécuter pour résoudre le problème initial;
  - Sémantique séquentielle
    - Toute instruction ne peut commencer que lorsque la précédente est terminée et son résultat disponible ;
  - ORDRE TOTAL dans l'exécution des différentes instructions;



# Programmation séquentielle et parallèle

---

- Programmation parallèle :
  - Plusieurs flots d'exécution (instructions + données);
  - Plusieurs instructions exécutées simultanément;
  - Plusieurs processeurs (ou cœurs) ;
  - Met en évidence les dépendances réelles entre instructions :
    - la tâche T2 dépend de la tâche T1 si et seulement si T2 a besoin du résultat de T1 (pour que le calcul soit juste)
    - si T2 ne dépend pas de T1 et T1 ne dépend pas de T2, alors T1 et T2 sont des tâches indépendantes
    - => Deux tâches indépendantes peuvent être exécutées dans un ordre quelconque, voire simultanément (i.e. en parallèle)

# Graphe de dépendance

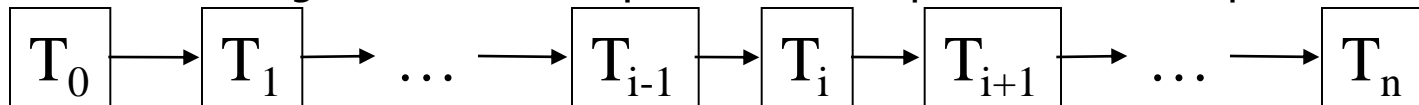
- Graphe de dépendance : met en évidence les relations de dépendances entre les tâches pour mener à bien une action;

- $T_1 \rightarrow T_2$  signifie T2 dépend de T1;

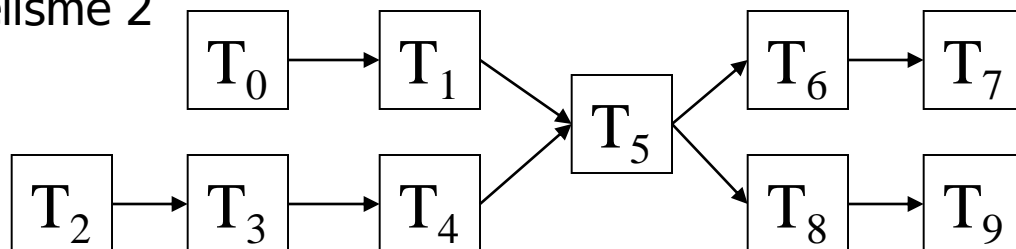
- Profondeur du graphe donne la dépendance ;

- Largeur du graphe donne l'indépendance (parallélisme) ;

- Programmation séquentielle : dépendance n et parallélisme 1



- Ex programmation parallèle avec 10 tâches : dépendance 6 et parallélisme 2





# Concurrence

---

- Les exécutions des tâches parallèles sont :
  - simultanées :
  - ou alternées (i.e. une tâche est interrompue pour laisser place à une autre tâche, puis reprise ultérieurement) ;
  - ou bien les deux ;
- ➔ Problème : les tâches peuvent accéder à des données communes et les modifier (notion de section critique);
- ➔ Solution : il faut trouver des mécanismes pour assurer la cohérence des données (mécanismes de verrous et d'exclusions mutuelles);



# Communication

---

- Communication et synchronisation :
  - Pour assurer la cohérence d'un calcul, des tâches parallèles peuvent se donner des "rendez-vous" avant de poursuivre.
- Ces points de rendez-vous sont appelés points de synchronisation :
  - Si la synchronisation concerne toutes les tâches parallèles
    - → synchronisation globale ou collective
  - Si les tâches ont des espaces d'adressages différents (ex sur machine à mémoire distribuée),
    - → communications (échange d'informations entre tâches "cloisonnées" (au sens de la mémoire))
- Communications
  - synchronisation globale => communication globale ou collective
  - synchronisation entre 2 tâches => communication point à point



# Types de parallélisme

---

- Quelles sont les sources de parallélisme dans une application ?
- 3 sources :
  - parallélisme de contrôle (tâches)
  - parallélisme de flux (pipeline)
  - parallélisme de données



# Parallélisme de contrôle

---

- Idée : *"Faire plusieurs choses en même temps"*
- Constatation naturelle :
  - Une application est composée d'actions que l'on peut faire en même temps
  - Exemple : l'exécution d'une recette de cuisine avec plusieurs cuisiniers
- Exploitation du parallélisme de contrôle consiste à gérer les dépendances entre les actions d'une application pour obtenir une allocation des ressources de calcul aussi optimale que possible
  - Extraction de ce parallélisme à partir du graphe de dépendance : largeur du graphe
- En pratique, degré de parallélisme peu élevé et souvent complexe à mettre en place (ex : ILP dans les processeurs)
- Correspond aux modèles de programmation parallèle suivants :
  - MIMD (Multiple Instruction Multiple Data)
  - MPMD (Multiple Program Multiple Data) : les processeurs exécutent des programmes différents avec leurs propres données



# Parallélisme de flux

---

- Idée : *"Travailler à la chaîne"*
- Principe : mode de fonctionnement en Pipe-line
  - on dispose d'un flux de données, généralement similaire, sur lesquelles on doit effectuer une suite d'opérations en cascade
  - les ressources de calcul sont associées aux actions et chaînées de manière à ce que les résultats des actions effectuées au temps  $T$  soient passés au temps  $T+1$  au processeur suivant
  - Exemple : machine vectorielle
- Degré de parallélisme est fonction de la profondeur du pipe (nombre d'étages)
- Travail sur des données vectorielles :
  - les vecteurs doivent être long pour minimiser le coût du chargement du pipe
- Le flux de données doit être continu au maximum (chaque arrêt/reprise du flux provoque un déchargement / chargement du pipe)





# Parallélisme de données

---

- Idée : *"Répéter une action sur des données similaires"*
- Principe : on partage les données et non plus les tâches
  - exemple : corvée de pommes de terre à l'armée avec plusieurs appelés du contingent
- Degré de parallélisme potentiellement élevé car fonction de la taille des données
- Correspond au modèle de programmation parallèle SPMD (Single Program Multiple Data)
  - Tous les processeurs exécutent le même programme avec leurs données propres
  - Viable pour un grand nombre de données
  - Très efficace pour beaucoup d'algorithmes de calcul intensif (calcul scientifique, films d'animation) : par la suite, nous allons nous concentrer sur ce modèle de programmation



# Paradigmes de la programmation parallèle

---

- Indépendamment des architectures matérielles des machines, deux modèles de programmation parallèles se dégagent :
  - modèle de programmation à mémoire distribuée
  - modèle de programmation à mémoire partagée
- En théorie, chaque modèle peut s'implémenter sur n'importe quel type d'architecture avec des effets collatéraux plus ou moins importants sur les performances



# Modèle de programmation à mémoire distribuée

---

- Conditions :
  - les tâches parallèles travaillent sur des mémoires distinctes, invisibles les unes des autres
  - les données (tableaux, etc...) sont éclatées (on parle de données distribuées) sur les différentes tâches parallèles
- Conséquence : pour assurer la justesse du résultat final, des communications inter-tâches deviennent obligatoires
- On parle alors de programmation par passage de messages (*Message Passing*)
- Adapté au modèle SPMD



# Modèle de programmation à mémoire distribuée

---

- Implémentation sur architecture à mémoire distribuée
  - facile car en reprend le principe :
    - sur une machine à mémoire distribuée, des processus qui ont leurs propres espaces d'adressage doivent envoyer des messages par le réseau pour échanger des infos
- Implémentation sur architecture à mémoire partagée
  - guère plus difficile :
    - par le biais de plusieurs processus en utilisant les segments de mémoire partagée pour les communications
    - par le biais d'un processus multithread en utilisant la mémoire de celui-ci pour échanger les informations entre threads



# Modèle de programmation à mémoire distribuée

---

- Encapsulation des échanges de messages, implémentées par des bibliothèques, comme par exemple :
  - PVM (Parallel Virtual Machine) : une des premières bibliothèques portables d'échanges de messages
  - MPI (Message Passing Interface) : le standard à l'heure actuelle, très répandue, issue de la collaboration d'industriels et d'universitaires
    - par la suite, nous allons étudier cette interface



# Modèle de programmation à mémoire partagée

---

- Condition :
  - les tâches parallèles ont une visibilité commune de la mémoire
- Conséquence :
  - il faut gérer les accès concurrents à la mémoire (section critique - "ne pas se marcher sur les pieds")
- Malgré la simplicité apparente de programmation, les performances peuvent être rapidement dégradées car :
  - les sections critiques ne sont pas parallèles (par définition)
  - on ne se rend pas compte de la localité des données et de la hiérarchie de la mémoire (pour les nœuds NUMA)



# Modèle de programmation à mémoire partagée

---

- Implémentation sur architecture à mémoire distribuée
  - difficile : comment "voir" la totalité de la mémoire ?
    - DSM (Distributed Shared Memory) : mécanisme logiciel permettant de donner l'illusion d'une mémoire unique à une mémoire physiquement distribuée
    - peut très vite couter cher car on ne se rend pas compte de l'accès distant aux données
- Implémentation sur architecture à mémoire partagée
  - naturelle car elle en reprend les principes :
    - dans un processus multithread, tous les threads ont accès à la mémoire du processus (qui peut adresser toute la mémoire du neoud)



# Modèle de programmation à mémoire partagée

---

- API POSIX pthread :
  - manipulation normalisée (POSIX) de threads au sein d'un processus
  - adapté pour le modèle MPMD
- OpenMP :
  - manipulation de threads par directive de compilation
- A l'heure actuelle, des outils implémentant le modèle à mémoire partagée font part d'une grande réflexion afin que les applications puissent exploiter au mieux les processeurs multicores
  - TBB, Cilk++, ABB, ...





# Résumé

---

- La programmation parallèle utilise les dépendances entre les tâches. Elle fait apparaître de nouvelles notions (concurrency, communication), mais également de nouvelles difficultés (debugging, performances);
- Les 3 sources de parallélisme sont les parallélismes de contrôle, de flux, et de données (ce dernier offrant le degré potentiel de parallélisme le plus élevé);
- 2 modèles de programmation parallèle se dégagent :  
programmations à mémoire distribuée, et à mémoire partagée.  
Le modèle à mémoire distribuée SPMD (parallélisme de données) est le plus répandu dans le milieu du HPC.