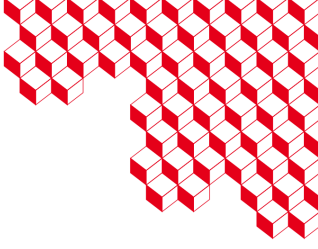


Kokkos/C++ training

Performance portability

Pierre Kestener

December 9-11th, 2024, Cerfacs, Toulouse

Schedule

Monday, December 9th, 2023 : Kokkos tutorial

- **Introduction to performance portability**, hardware trends, programing models for heterogeneous computing platforms
- GPU Computing / Cuda refresher
- supercomputer `calypso` : a CPU (Nvidia Grace ARM) + GPU (Nvidia Hopper H100) computing platform : short overview
- Hands-on 00 : Get familiar with `calypso` compile environment, use `saxpy` to review CPU/GPU differences
- **C++ Kokkos : features overview**
- Hands-on 01 : **retrieve Kokkos sources**, how to build with `Make` or `cmake`, explore different hardware/compiler configurations
- **Kokkos abstract concepts overview** : parallel programing patterns, data container
- Hands-on 02 : **build a Kokkos app**, how to build with `Make` or `cmake`, how to integrate `kokkos` in an existing application

Schedule

Tuesday, December 10th, 2023 : Kokkos tutorial

- **Kokkos abstract concepts overview (continue)** : parallel programing patterns, data container
- Hands-On 03 : Revisit simple example **SAXPY**
  $\Rightarrow$ simplest computing kernel in Kokkos, performance measurement
- Hands-On 04 : Simple example **Mandelbrot set**
  $\Rightarrow$ 1D Kokkos ::View + linearized index (+ asynchronous execution)
- **a Kokkos miniapp skeleton project with cmake**
- Hands-On 05 : Simple examples **Stencil + Finite Difference**
  $\Rightarrow$ 2D Kokkos ::View
- Hands-On 05-b : **Laplace exercise**
  $\Rightarrow$ pure Kokkos versus Kokkos + MPI + hwloc (multiGPU)
- Hands-On 06 : **Illustrate how to use random number generator in kokkos**
  $\Rightarrow$ RNG 101, parallel compute π with Monte Carlo
- Hands-On 07 : $\Rightarrow$ use Kokkos lambda and hierarchical parallelism (Team Policy)

Schedule

Wednesday, December 11th, 2023 : Kokkos tutorial

- Hands-On 08 : : Kokkos and simd
- Hands-On 09 : : Kokkos ecosystem python bindings and code generation : pykokkos
- Hands-On 10 : : Kokkos ecosystem FLCL (Fortran Language Compatibility Layer)
- Hands-On 11 : : Kokkos ecosystem for linear algebra : kokkos-kernels
- introduction to `Kokkos::tools` (integration with profiling tools, e.g. Nvidia `nsys`)
- using Kokkos and MPI. Using a simple MPI+Kokkos CFD miniApp : **Euler solver**
  $\Rightarrow$ performance measurement for several Kokkos backends (OpenMP, CUDA)

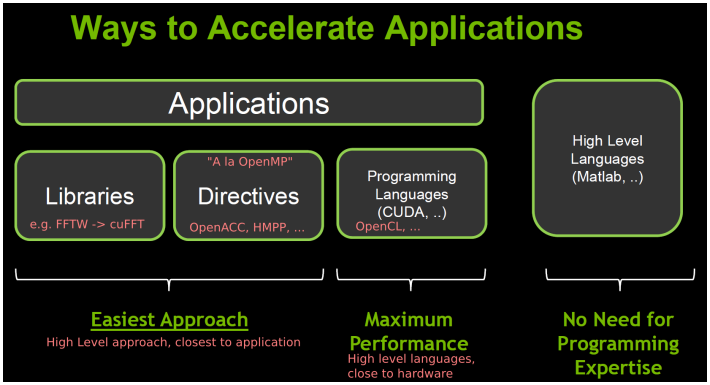
1. General Introduction

Content

- **Quick reviewing main HPC architectures (hardware and software) trends** : multicore, manycore, GPU, ARM, ...
- **What is performance portability ?**
- **A good software abstraction / programing model(s) (?)**
  - **library, framework, programming models ?**
  - **Parallel programming patterns**
  - **Native language, directives, DSL ?**
- **Introduction to Kokkos : a C++ library for (aiming at) performance portability**
  - Node-level parallelism / shared memory parallelism,
  - parallel alogithmic patterns,
  - data containers.

From low-level native to high-level programmning

Revisiting ways to **develop software applications** not only for accelerators, but multiple architectures



reference : Axel Koehler, NVIDIA, 2012

Find a good trade-off between *ease of approach* and *good performance* on **multiple architectures**.

Exascale race : technological challenge(s) - hardware trends

Exascale is about...

- ... reaching exaflops $= 10^{18}$ flop/s
  - 1.3 exaflops of aggregate peak flops
  - $\sim$ 8 PBytes of RAM
- **Building a computing ecosystem :** from applications, system software, hardware technologies, and architectures
- Doing usefull work : HPC simulation ? AI ? both !
- **2022, first exascale machine named** frontier

Many challenges :

- **Most challenging constraint :** fitting the **electrical power envelop** $P \in [20 - 40]$ MW
- develop new applications / adapt old ones
- system software, application software stacks
- hardware technologies (**interconnect, storage, processor architectures,...**)

Frontier (AMD CPU and GPU)

current #1 @top500

Fugaku (Fujitsu, ARM/A64FX) (Japan, 2020)

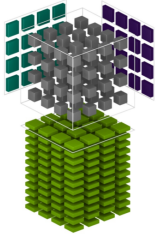
current #2 @top500

Exascale race : technological challenge(s) - hardware trends

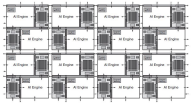
...also about building an economic ecosystem

- New architectures designed to address **several markets** : HPC, AI, IoT, near-sensor computing, automotive, ...
- **Hardware vendors already designing/optimizing new architectures for AI** (always back and forth between general purpose and application specific) :e.g
  - Nvidia (Tensor Core),
  - Xilinx (Alveo / Versal),
  - Intel (e.g. Bfloat16),
- **Semantic shift : HPC = simulation + AI**
- **Cost of designing a new chip skyrocketting, ...**

Nvidia Tensor Core, Volta (2017)



Xilinx AI Engine array (2019)



(Pre-)Exascale machines - (hybrid) architecture diversity !

- **US** : Frontier, Summit , Sierra ⇒ mostly GPUs (AMD / Nvidia), GPU-based architecture, #1, #5 and #6 @top500 ; **first exascale machine(s)**
  - Frontier (Oak Ridge NL, 2022) : AMD EPYC + Radeon Instinct GPU MI250X
  - Aurora (Argonne NL, end of 2023) : Intel Xe GPU
- **China :** 3 machines
  - Phytium FT2000/64 ARM chips + Matrix2000 GPDSP accelerators ⇒ #4 @top500, Tianhe-2A, 61 PFlops
  - 260-core Shenwei, **homegrow technology** hardware + software (C++/fortran compiler + OpenACC) ⇒ #7 @top500 (June 2023) , Sunway TaihuLight, 93 PFlops
  - Dhyana, AMD-licenced x86 multicore (300 M$!), identical to AMD EPYC
- **Japan : Fugaku**(Fujitsu, ARM, RIKEN) A64FX ARM (**home grown**, started in 2014, **#2 @top500 (June 2022)**, 900 M$), GPU, etc ...
- **Europe** : new organization EuroHPC (2019), EC H2020 budget (∼ 500 M€)
  home grown (EPI) ARM and RISC-V architecture, early stage

(Pre-)Exascale machines in Europe

FOR EXASCALE = HYBRID ARCHITECTURES IS A GENERAL TREND

In Europe, US or Japan (but not in Japon – 40MW, 0.5EF)

428 PF peak, 309 PF (LUMI-G),
4096 compute nodes: 62% hybrid,
38% scalar (**#3 top500**)

Leonardo
EVIDEN
an atos business

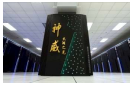
304 PF peak, 238 PF HPL
4992 compute nodes:
69% hybrid, 31% scalar (**#4 top500**)

Marenostrum 5 (314 PF peak) to come with similar ratios EVIDEN an atos business

1.68 EF peak, 1.1 EF HPL (**#1 top500**)
9472 nodes, 100% hybrid, 21MW

>2 EF peak, > 9000 nodes, 100% hybrid (to come in November ?)

>2 EF peak
end 2023

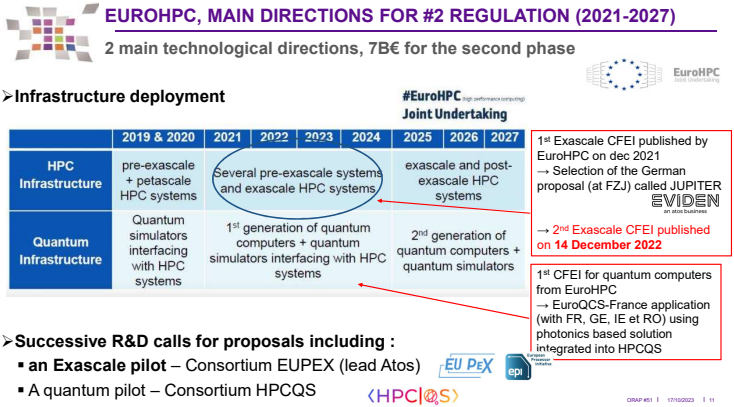


Sunway OceanLight
1.3 EF peak, 1.05 EF HPL (?)
>107k nodes, 42M scalar cores but 35MW (!)

NUDT Tianhe-3
1.7 EF peak, 1.3 EF HPL (?)
> Phytium 2000+ ARM + Matrix 2000+ MTP accelerators

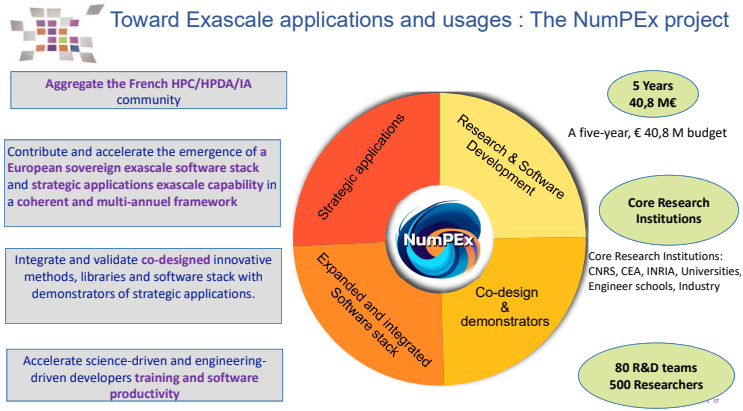
source : http://orap.irisa.fr/wp-content/uploads/2023/10/01-GENCI-ORAP51.pdf

(Pre-)Exascale machines in Europe



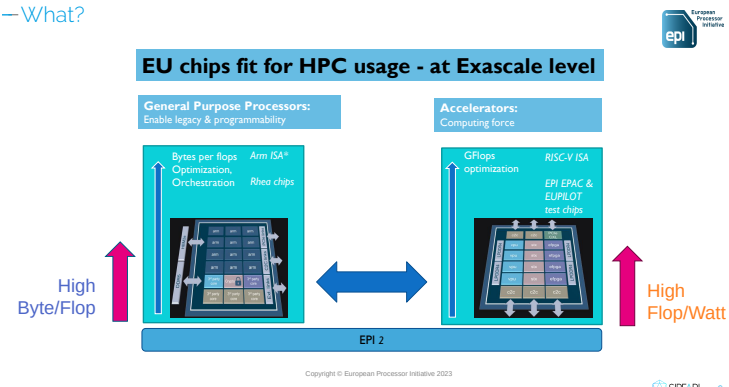
source : http://orap.irisa.fr/wp-content/uploads/2023/10/01-GENCI-ORAP51.pdf

(Pre-)Exascale machines in Europe



source : http://orap.irisa.fr/wp-content/uploads/2023/10/01-GENCI-ORAP51.pdf

European Processor Initiative (EPI)

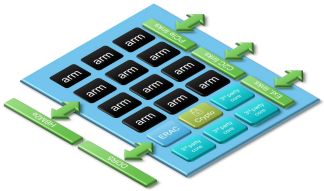


source : http://orap.irisa.fr/wp-content/uploads/2023/10/06-SIPearl-ORAP51.pdf

European Processor Initiative (EPI)

Rhea Key Features

Element	Features
Core	- Arm ISA - **Neoverse V1 cores** - **SVE 256** per core supporting 64/32/BF16 and Int8 - ArmVirtualization extensions
SoC	- NoC: Arm mesh fabric - Advanced RAS including Arm RAS extensions - Link protection for NoC & high-speed IO - ECC support for selected memory
Cache	- **Large L3** (Shared Level Cache, **SLC**) - RAS supported for all cache levels
Memory	- **HBM2e** - **And DDR5** - ECC for memory and link protection for controllers
High Speed I/O	- **PCIe, CCIX & CXL** - Root and endpoint support
Other I/O	- USB, GPIO, SPI, I^2C
Power Management	- Power management block to optimize perf/Watt accross use cases and workloads.
Security Block Support	- Secure boot and secure upgrade - Crypto - True random number generation



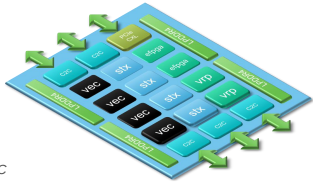
Design targets:
- High Byte/Flop ratio
- Compute performance and efficiency for real-applications

source : http://orap.irisa.fr/wp-content/uploads/2023/10/06-SIPearl-ORAP51.pdf

European Processor Initiative (EPI)

EPAC Vision and Contributions

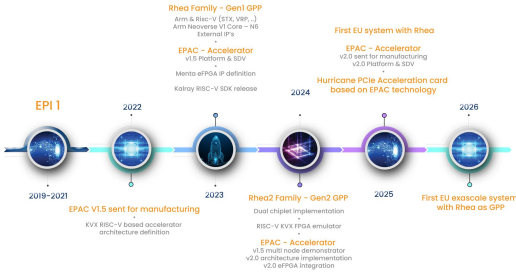
- **VEC -** Self-hosted RISC-V CPU + wide VPU (256 double elements) supporting RVV 0.7.1 / 1.0
- **STX -** RISC-V CPU + specific cores for stencil and neural network computation
- **VRP -** RISC-V CPU with support for variable precision arithmetic (data size up to 512 bit)
- **eFPGA -** On-chip reconfigurable logic
- **Ziptillion -** IP compressing/decompressing data to/from the main memory
- ***KVX -*** *FPGA demonstrator of the Kalray RISC-V CPU targeting HPC and ML*



source : http://orap.irisa.fr/wp-content/uploads/2023/10/06-SIPearl-ORAP51.pdf

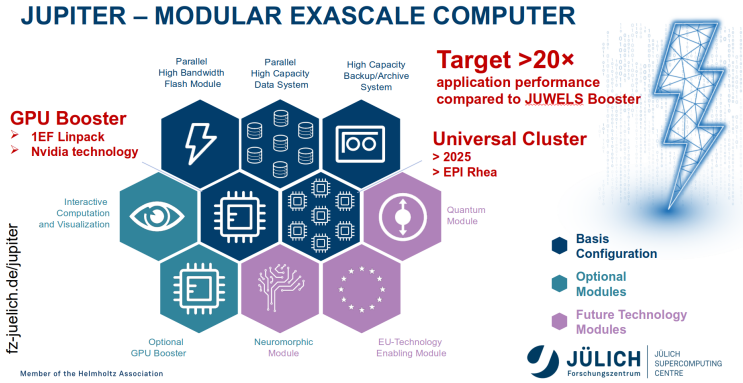
European Processor Initiative (EPI)

EPI2 timeline



source : http://orap.irisa.fr/wp-content/uploads/2023/10/06-SIPearl-ORAP51.pdf

Jupiter : first European exascale supercomputer in 2024



source : https://numpex.irisa.fr/wp-content/uploads/2023/10/InPEx_Pre-workshop_Jupiter-presentation.pptx (NumPEx, pre-workshop, October 2023)

Supercomputers architectures - TOP500

A Supercomputer is designed to be at bleeding edge of current technology.
Leading technology paths (to exascale) using TOP500 ranks (**June 2023**)

- **Massively Multithread / GPU :** (# 1, 3, 4, 5, 6, 8 and 9)
- **Multicore :** Only two in the 10 largest, Fugaku (ARM64FX, #2), Tianhe-2A (Intel Xeon, X86-64, #10)
- **Manycore/Sunway TaihuLight** (SW26010, # 7)

Sunway Taihulight : programmed with MPI+OpenACC

HPC architectures - Trends - Who's driving ?

- Relatively new chip makers :
  - Amazon is using its own ARM Graviton3 processor ;
  - Microsoft and Google are also designing chips with AI app in mind
- AI (NN training and inference) is driving lots of computing power in data center and clouds ; Intel has 2 different hardware lines
  - Intel Gaudi 2 [1] is an *AI accelerator*, aimed at competing with Nvidia or AMD GPU
  - Intel Xe for regular HPC, can also do AI (tensorFlow / PyTorch)
- Still on-going strong competition among Nvidia, AMD and Intel on the GPU market (high-end hardware, as medium range hardware)
- *traditional* HPC market has become (much) smaller than AI todays.

1. https://habana.ai/wp-content/uploads/2023/10/Intel-Gaudi2-AI-Accelerators-whitepaper.pdf

Multiples levels of hierarchy :

- Need to aggregate the computing power of several 10 000 nodes !
- network efficiency : latency, bandwidth, topology
- memory : on-chip (cache), out-of-chip (DRAM), IO (disk)
- emmerging **hybrid programming model : MPI + X**
- What is **X** ? OpenMP, OpenAcc, ..., Kokkos, RAJA, ...
- Even at node level MPI+X is required : e.g. KNL

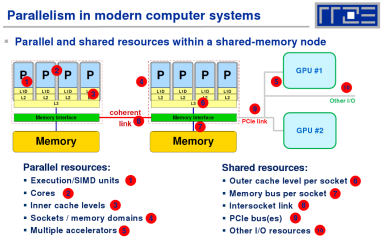
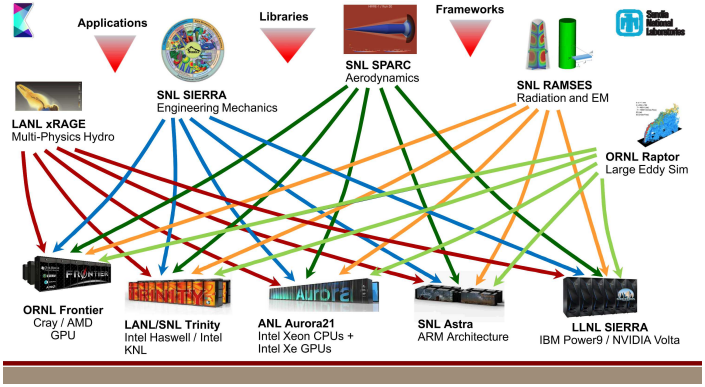


Figure 1 – Multi-core node summary, source : multicore tutorial (SC12) by G. Hager and G. Wellein

2. What is Performance Portability ?

Why do we need performance portable programming models ?



source : Jeff Miles, Evolution of the Kokkos ecosystem, 2020

Why do we need performance portable programming models ?

The HPC Hardware Landscape

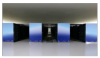
Current Generation: Programming Models OpenMP 3, CUDA and OpenACC depending on machine

LANL/SNL Trinity
Intel Haswell / Intel KNL
OpenMP 3

LLNL SIERRA
IBM Power9 / NVIDIA Volta
CUDA / *OpenMP*[(a)]

ORNL Summit
IBM Power9 / NVIDIA Volta
CUDA / *OpenACC* / *OpenMP*[(a)]

SNL Astra
ARM CPUs
OpenMP 3



Riken Fugaku
ARM CPUs with SVE
OpenMP 3 / *OpenACC*[(b)]

Upcoming Generation: Programming Models OpenMP 5, CUDA, HIP and DPC++ depending on machine

NERSC Perlmutter
AMD CPU / NVIDIA GPU
CUDA / *OpenMP 5*[(c)]

ORNL Frontier
AMD CPU / AMD GPU
HIP / *OpenMP 5*[(d)]

ANL Aurora
Xeon CPUs / Intel GPUs
DPC++ / *OpenMP 5*[(e)]

LLNL El Capitan
AMD CPU / AMD GPU
HIP / *OpenMP 5*[(d)]

(a) Initially not working. Now more robust for Fortran than C++, but getting better.
(b) Research effort.
(c) OpenMP 5 by NVIDIA.
(d) OpenMP 5 by HPE.
(e) OpenMP 5 by Intel.

July 17, 2020 8/72

source : Kokkos Lectures, Module 1, 2020

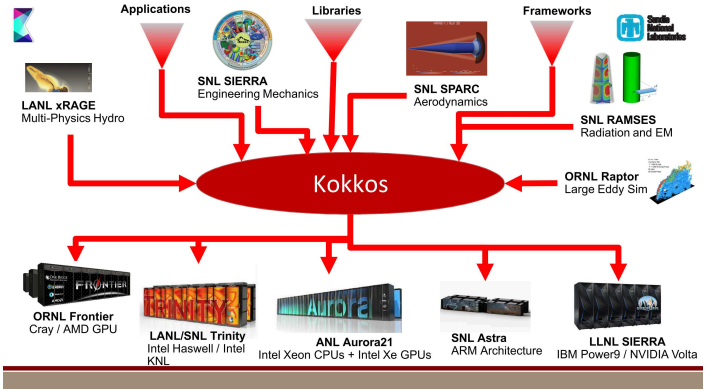
Cost of Porting Code

10 LOC / hour ~ 20k LOC / year

- Optimistic estimate: 10% of an application is modified to adopt an on-node Parallel Programming Model
- Typical Apps: 300k – 600k Lines
  - 500k x 10% => Typical App Port 2.5 Person-Years
- Large Scientific Libraries
  - E3SM: 1,000k Lines x 10% => 5 Person-Years
  - Trilinos: 4,000k Lines x 10% => 20 Person-Years

source : Jeff Miles, Evolution of the Kokkos ecosystem, 2020

Why do we need performance portable programming models ?



source : Jeff Miles, Evolution of the Kokkos ecosystem, 2020

A Brief History of Kokkos

- Beginnings (2003-2008) ***version 0.x***
  - Sparse kernels for vector-matrix operations (Kokkos-Kernels predecessor)
- Node abstractions (2008-2010) ***version 0.5***
  - Multi-core and heterogeneous hardware drives need for abstraction
  - KokkosNode concept used to abstract execution node
- Complete programming model (2010-2014) ***version 1.0***
  - Data abstractions combined with execution abstractions
  - Primary programming model for SNL ATDM applications
  - Task DAG added

source : Jeff Miles, Evolution of the Kokkos ecosystem, 2020

A Brief History of Kokkos (cont.)

- Externally available product (2015-2018) ***version 2.0***
  - Open source with regular maintenance
  - Kokkos Kernels re-invigoration on top of programming model
  - Regular training with bootcamps
  - Tools added for debugging and profiling
- C++ and exascale platforms (2019-Present) ***version 3.0***
  - C++ Standards with abstract parallel concepts
  - Features to meet growing needs of ASC applications
  - Backend infrastructure to prepare for exascale systems
  - Tools beyond profiling
  - Capabilities of Kokkos Kernels expanded

source : Jeff Miles, Evolution of the Kokkos ecosystem, 2020

How pervasive is Kokkos today?

- Production Code Running Real Analysis Today
  - We got about **12** or so
- Production Code or Library committed to using Kokkos and actively porting
  - Somewhere around **35**
- Packages In Large Collections (e.g. Tpetra, MueLu in Trilinos) committed to using Kokkos and actively porting
  - Somewhere around **65**
- Counting also proxy-apps and projects which are evaluating Kokkos (e.g. projects who attended boot camps and trainings).
  - Estimate **100-150** packages.

At least 30 ECP products depend on Kokkos (14 critical dependencies)

source : Jeff Miles, Evolution of the Kokkos ecosystem, 2020

Performance portability

- **Developing / maintaining** a **separate** implementation of an application for each **new hardware platform** (Intel KNL, Nvidia GPU, ARMv8, ...) is **less and less realistic**
- **Identical code** will never perform **optimally** on all platforms [2]
- Is it possible to have a **single set of source codes** that can be compiled for different hardware targets ?
- Performance portability should be understood as a single source code base with
  - **good** performance on different architectures
  - a relatively **small amount of effort** required to tune app performance from one architecture to another.

  source http://www.nersc.gov/research-and-development/application-readiness-across-doe-labs

2. source : Matt Norman, WACCPD 2016

Performance portability

- **Developing / maintaining** a **separate** implementation of an application for each **new hardware platform** (Intel KNL, Nvidia GPU, ARMv8, ...) is **less and less realistic**
- **Identical code** will never perform **optimally** on all platforms [2]
- Is it possible to have a **single set of source codes** that can be compiled for different hardware targets ?
- **performance portability** is achieved when software implementation
  - compiles and runs on **multiple architectures**,
  - obtains performant **memory access patterns** across architectures,
  - can **leverage architecture-specific features** where possible.

2. source : Matt Norman, WACCPD 2016

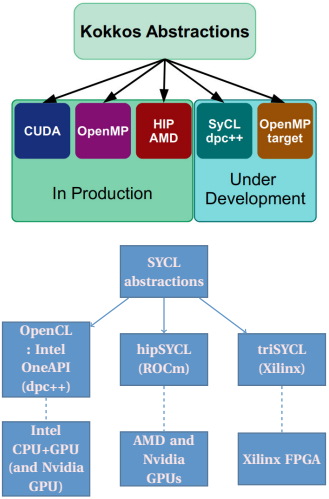
Performance portability issue

Developper productivity versus optimization

- **Taking into account hardware details is a hard job**
  - CPU vector length : 256 bits (8 *vector threads*)
    Heavily cache-based
  - KNL vector length : 512 bits x 2 (16-32 *vector threads*)
    Moderately cache-based, some latency/bandwidth hiding
  - GPU vector length : 65 536 bit (2048 *GPU vector threads*)
    Less cache-based, heavy on latency/bandwidth hiding
- **Find ways of writing codes that avoid optimization blockers**
  $\Rightarrow$ Constructs that can be used to express operations without going into details

Performance portability issue : algorithmic patterns

- Is it possible to have a single set of source codes that can be compiled for different hardware targets ?
- **Low-level native language :** CUDA, HIP, OpenCL, ...
- **Directive approach (code annotations)** for multicore/GPU, ... :
  - OpenMP $\geq$ 4.5 (nvhpc, Clang, GNU, ...)
  - OpenACC $\geq$ 2.6 (nvhpc, Cray, GNU, ...)
- **Other high-level library-based approaches** (mostly c++-based, à la TBB) :
  - Some provide STL-like algorithmics patterns (e.g. Thrust is CUDA-based with backends for other archs, lift, arrayFire (numerical libraries, language wrappers, ...))
  - Kokkos, RAJA, OCCA, Alpaka, Celerity-Runtime, hpx, Dash-project, agency, ...
  - SYCL (Khronos Group *standard*), C++ high-level layer on top of OpenCL. Intel OneAPI/DPCPP (Intel CPU/GPU/FPGA, Nvidia GPUs), CodePlay, AMD and Nvidia GPUs, Keryell/Xilinx
  - **C++-17 built-in parallelism for multicore and GPUs**, e.g. :
    - Nvidia's hpc-sdk (May 2020)
    - Intel OneAPI/TBB



Performance portability issue : memory management

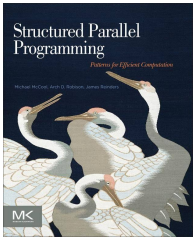
- Right now **directives-based approaches** focus on algorithmic pattern, and less on memory layout (might change in the near future, at least in OpenMP).
- CPU and GPU for example require **different memory layout** for **maximun performance :**
  - vectorization on CPU
  - memory coalescence on GPU
- Some libraries like Kokkos promote **memory layout** as a **major concern**

Motivation: Variety in Memory Hierarchies

Platform	Memory Kind							
	Constant	Texture	SPM	DDR	eDRAM	GDDR	HBM	NVRAM
Intel® Xeon® Processor	-	-	-	✔	-	-	-	-
Intel® Xeon Phi™ Coprocessor	-	-	-	-	-	✔	-	-
Intel® Xeon Phi™ Processor	-	-	-	✔	-	-	✔	-
Future System w/ 3D XPoint™ Technology	-	-	-	✔	-	-	-	✔
Intel® HD Graphics	-	-	✔	✔	✔	-	-	-
Intel® Iris™ Graphics	-	-	✔	✔	✔	-	-	-
Current Generation NVIDIA* GPU	✔	✔	✔	-	-	✔	-	-
Future Generation NVIDIA* GPU	✔	✔	✔	-	-	✔	✔	-

Programming with structured parallel patterns

- **pattern** : **a basic structural entity of an algorithm**
- book Structured Parallel Programming : Patterns for Efficient Computation

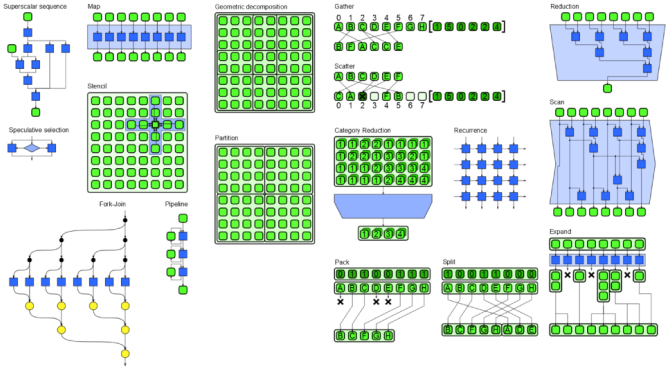


- implementation : Intel TBB, OpenMP, OpenACC and many others
- OpenMP/OpenAcc for GPU/XeonPhi : pattern-based comparison : map, stencil, reduce, scan, fork-join, superscalar sequence, parallel update

reference :

A Pattern-Based Comparison of OpenACC and OpenMP for Accelerator Computing

Parallel Patterns: Overview



reference : Structured Parallel Programming with Patterns, SC13 tutorial, by M. Hebenstreilt, J. Reinders, A. Robison, M. McCool

Active/Passive library and accelerator programming

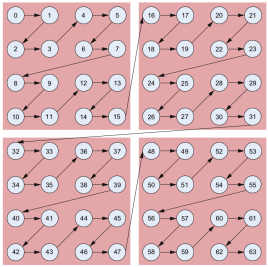
- **passive libraries :** a collection of subroutines
- **active libraires :** take an active role in compilation (specialize algorithms, tune themselves for target architecture).

Library	CUDA	OpenCL	Other	Type
Thrust	X		OMP, TBB	header
Bolt		X	TBB, DX11	link
VexCL	X	X		header
Boost.Compute		X		header
C++ AMP		X	DX11	compiler
SyCL		X		compiler
ViennaCL	X	X	OMP	header
SkePU	X	X	OMP, seq	header
SkelCL		X		link
HPL		X		link
CLOGS		X		link
ArrayFire	X	X		link
CLOGS		X		link
hemi	X			header
MTL4	X			header
Kokkos	X		OMP, PTH	link
Aura	X	X		header

reference : The Future of Accelerator Programming in C++, S. Schaetz, May 2014

Complex memory layout for performance

- How to improve **space (memory) locality** in algorithm implementations ?
- ***High Performance Parallelism Pearls***, **Morton order to improve memory locality**, by Kerry Evans (INTEL), chap. 28
- **matrix transpose**, **dense matrix multiplication** on Xeon, KNC
- Same feature used in some Adaptive Mesh Refinement PDE solver.



dim	naïve 32 thr	Morton 32 thr	speedup
256	0.1	0.188	0.53
512	0.05	0.169	0.29
1024	0.62	0.366	1.7
2048	2.89	0.871	3.3
4096	211	7.92	26.6
8192	1850	60.2	30.7
16384	13695	473	29.0
32768	Too long	3989	--

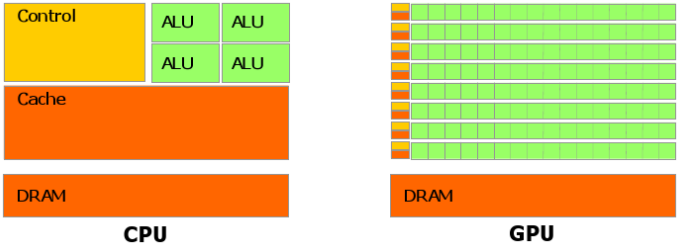
dim	naïve 244 thr	Morton 244 thr	speedup
256	0.02	0.003	6.67
512	0.26	0.008	32.5
1024	1.78	0.046	38.7
2048	12.87	0.4	32.18
4096	105.5	2.9	36.38
8192	105.5	23	36.74
16384	6597	181	36.4
32768	Too long	1468	--

CPU versus GPU : hardware and software comparison

- What are the major differences between CPUs and GPUs in terms of hardware architecture ?
- What is the meaning of *latency-oriented architecture* ?
- What is the meaning of *throughput-oriented architecture* ?

Main differences between CPU and GPU (CUDA refresher)

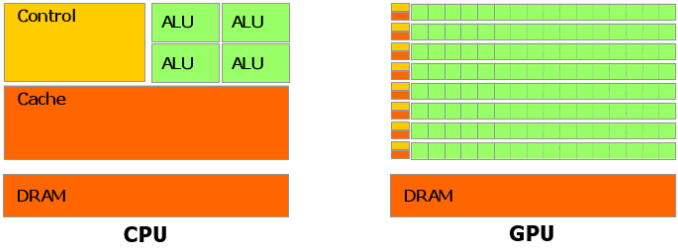
Architecture design differences between manycore GPUs and general purpose multicore CPU ?



- **Different goals produce different designs :**
  - **CPU** must be good at everything, parallel or not
  - **GPU** assumes work load is highly parallel

Main differences between CPU and GPU (CUDA refresher)

Architecture design differences between manycore GPUs and general purpose multicore CPU ?



- **CPU** design goal : optimize architecture for sequential code performance : minimize latency experienced by **1 thread**
  - sophisticated (i.e. large chip area) control logic for instruction-level parallelism (branch prediction, out-of-order instruction, etc...)
  - CPU have large cache memory to reduce the instruction and data access latency

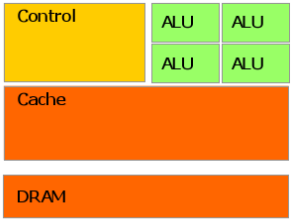
Main differences between CPU and GPU (CUDA refresher)

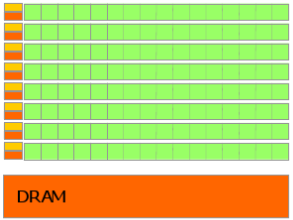
Architecture design differences between manycore GPUs and general purpose multicore CPU ?

- **GPU** design goal : maximize throughput of **all threads**
  - # threads in flight limited by resources => lots of resources (registers, bandwidth, etc.)
  - multithreading can hide latency => skip the big caches
  - share control logic across many threads

Main differences between CPU and GPU (CUDA refresher)

Architecture design differences between manycore GPUs and general purpose multicore CPU ?

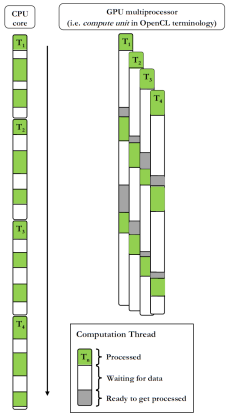




- **CPU (latency oriented architecture)**
  - **1 thread $\Leftrightarrow$ 1 core**
- **GPU (througput oriented architecture)**
  - **# threads $\gg$ # cores**

Main differences between CPU and GPU (CUDA refresher)

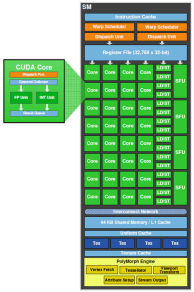
Architecture design differences between manycore GPUs and general purpose multicore CPU ?



- **CPU : Minimazing memory latency** by having large cache memory.
- **GPU : Hiding memory latency** by having a large number of thread per core.

Nvidia Fermi hardware (2010)

- **Streaming Multiprocessor** (32 cores), hardware control, queuing system
- **GPU = scalable array of SM** (up to 16 on Fermi)
- **warp : vector of 32 threads**, executes the same instruction in lock-step
- **throughput limiters** : finite limit on warp count, on register file, on shared memory, etc...



Streaming Multiprocessor (SM)

Nvidia Fermi hardware (2010)

CUDA Hardware (HW) key concepts

- Hardware thread management
  - HW thread launch and monitoring
  - HW thread switching
  - up to 10 000's lightweight threads
- SIMT execution model
- Multiple memory scopes
  - Per-thread private memory : (register)
  - Per-thread-block shared memory
  - Global memory
- Using threads to hide memory latency
- Coarse grained thread synchronization

From multi-core CPU to manycore GPU

Comparing brut performance (FLOPS and Bandwidth) for CPU and GPU :

Nvidia A100 “Ampere” SXM4 specs

Architecture

- 54.2 B Transistors
- ~ 1.4 GHz clock speed
- ~ 108 “SM” units
  - 64 SP “cores” each (FMA)
  - 32 DP “cores” each (FMA)
  - 4 “Tensor Cores” each
  - 2:1 SP:DP performance
- 9.7 TFlop/s DP peak (FP64)
- 40 MiB L2 Cache
- 40 GB (5120-bit) HBM2
- MemBW ~ 1555 GB/s (theoretical)
- MemBW ~ 1400 GB/s (measured)

© Nvidia

$$P^{DP}_{peak} = n_{SM} \cdot n_{core} \cdot n_{FP} \cdot f$$

# SMs	# CUDA cores/SM	# FP ops/cy

$$n_{SM} = 108$$
$$n_{core} = 32$$
$$n_{FP} = 2\frac{\text{flops}}{\text{cy}}$$
$$f = 1.4\frac{\text{Gcy}}{\text{s}}$$

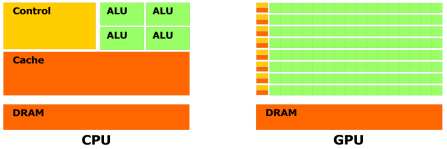
reference : Node-level performance engineering, training, HLRS, June 2023

From multi-core CPU to manycore GPU

Comparing brut performance (FLOPS and Bandwidth) for CPU and GPU :

Trading single thread performance for parallelism:
GPGPUs vs. CPUs

GPU vs. CPU
light speed estimate

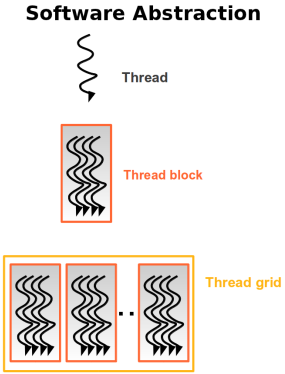


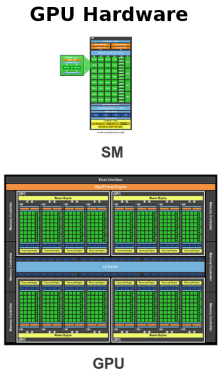
	2 x AMD EPYC 7742 "Rome"		NVidia Tesla A100 "Ampere"
Cores@Clock	2 x 64 @ 2.25 GHz		108 SMs @ ~1.4 GHz
FP32 Performance/core	72 GFlop/s		~179 GFlop/s
Threads@STREAM	~16		~ 100000
FP32 peak	9.2 TFlop/s	~2x	~19.5 TFlop/s
Stream BW (meas.)	2 x 180 GB/s	~4x	1400 GB/s
Transistors / TDP	~2x40 Billion / 2x225 W		54 Billion/400 W

reference : Node-level performance engineering, training, HLRS, June 2023

CUDA - connecting program and execution model

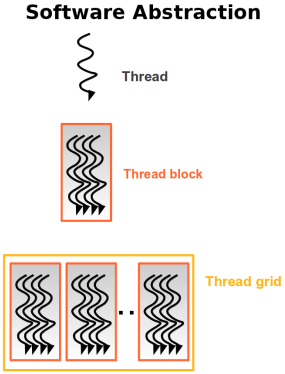
- **Need a programming model to efficiently use such hardware**; also provide **scalability**
- Provide a simple way of partitioning a computation into fixed-size blocks of threads

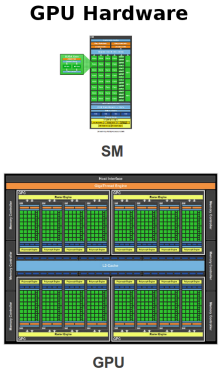




CUDA - connecting program and execution model

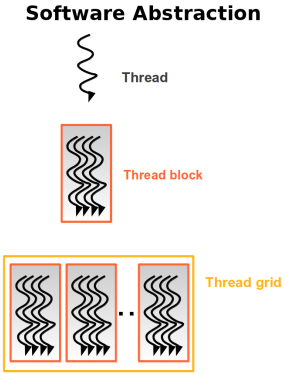
- **Total number of threads** must/need be quite larger than number of cores
- **Thread block** : **logical array of threads**, large number to hide latency
- **Thread block size** : control by program, specify at runtime, better be a multiple of warp size (i.e. 32)

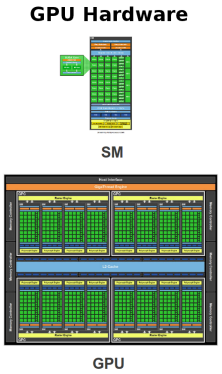




CUDA - connecting program and execution model

- **Must give the GPU enought work to do !** : if not enough thread blocks, some SM will remain idle
- **Thread grid** : **logical array of thread blocks** distribute work among SM, several blocks / SM
- **Thread grid** : chosen by program at runtime, can be the total number of thread / thread block size or a multiple of # SM

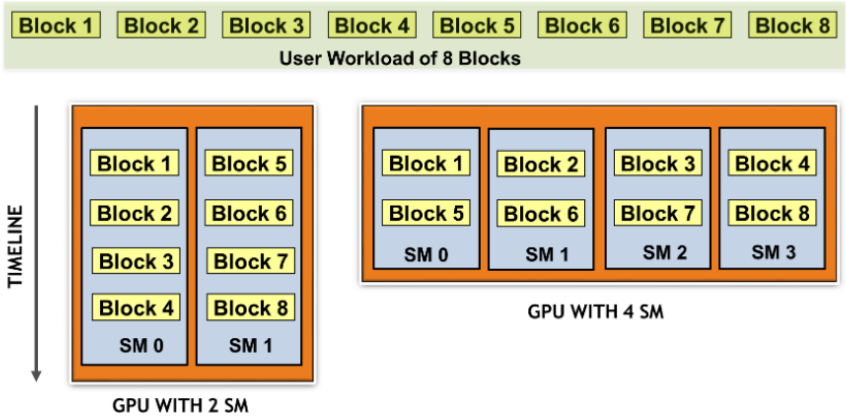




CUDA - connecting program and execution model

Why two levels of hierarchy block / grid ?

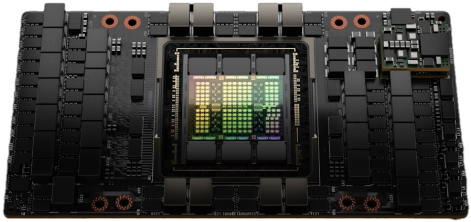
- scale/adapt to different GPUs (make code independent from hardware details, e.g. number of SM)



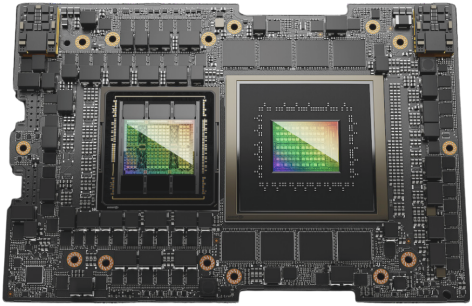
3. calypso computing platform

Nvidia Grace/Hopper superchip

H100



GH200



- CPU and GPU on the same board : very high CPU/GPU memory bandwidth, unified memory access

https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

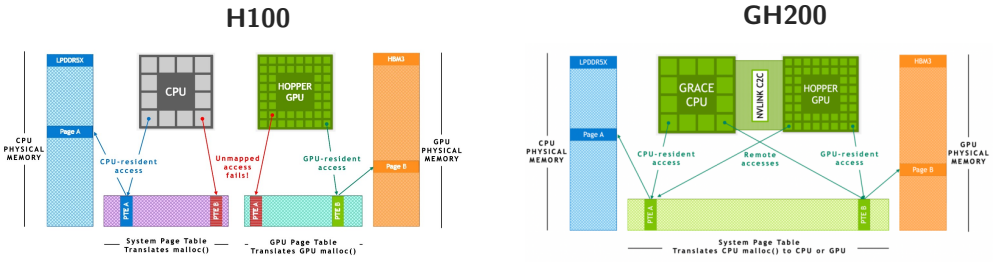
https://www.nvidia.com/fr-fr/data-center/grace-hopper-superchip/

https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper

https://developer.nvidia.com/blog/nvidia-grace-hopper-superchip-architecture-in-depth/

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#unified-memory-programming

Nvidia Grace/Hopper superchip



https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

https://www.nvidia.com/fr-fr/data-center/grace-hopper-superchip/

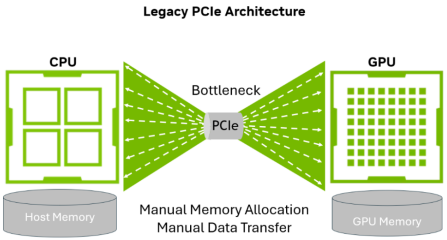
https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper

https://developer.nvidia.com/blog/nvidia-grace-hopper-superchip-architecture-in-depth/

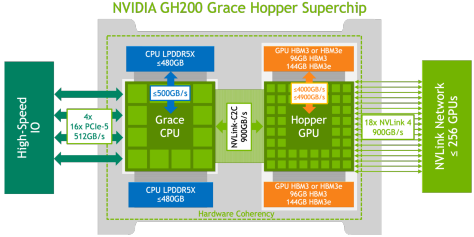
https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#unified-memory-programming

Nvidia Grace/Hopper superchip

H100



GH200



https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

https://www.nvidia.com/fr-fr/data-center/grace-hopper-superchip/

https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper

https://developer.nvidia.com/blog/nvidia-grace-hopper-superchip-architecture-in-depth/

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#unified-memory-programming

- **About calypso computing platform**
  - Minimal information about software environment, how to build and run an application, submit a job on a GPU node of calypso
  - **Examples of job submission scripts :** exercises/job.sh
  - How to run job ? sbatch ./job.sh
  - Here is an example job.sh :

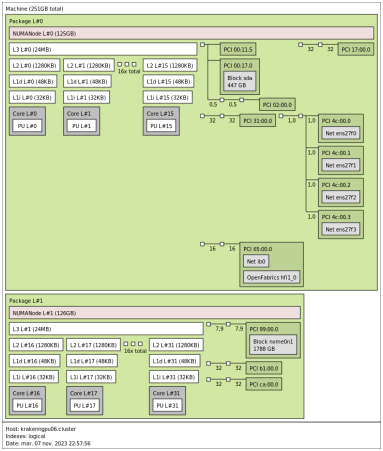
```
#!/bin/bash
#SBATCH -J compute_info
#SBATCH -N 1
#SBATCH --partition grace
#SBATCH --gres=gpu:1
#SBATCH --ntasks-per-node=1
#SBATCH --mail-user=me@somewhere.org
#SBATCH --mail-type=END
cd $SLURM_SUBMIT_DIR

lstopo -p lstopo_grace.xml
cat /proc/driver/nvidia/version
nvidia-smi
lscpu
```

- **About calypso computing platform**
  - Minimal information about software environment, how to build and run an application, submit a job on a GPU node of calypso

```
# Beware that calypso login nodes are x86-64 CPU, while Grace CPU is ARM
# ==> YOU NEED TO LOG ONTO A GRACE NODE TO COMPILE (cross-compilation not possible here)
#
# ask for an interactive access to a computing node on calpyso grace partition
[kestener@calypso-login2 ~]$ salloc -p grace --gres=gpu:0 -n 24
salloc: Pending job allocation 14614
salloc: job 14614 queued and waiting for resources
salloc: job 14614 has been allocated resources
salloc: Granted job allocation 14614
salloc: Waiting for resource configuration
salloc: Nodes calypso-grace01 are ready for job
[kestener@calypso-login2 ~]$ ssh grace01
[kestener@calypso-grace01 ~]$ nvidia-smi
```

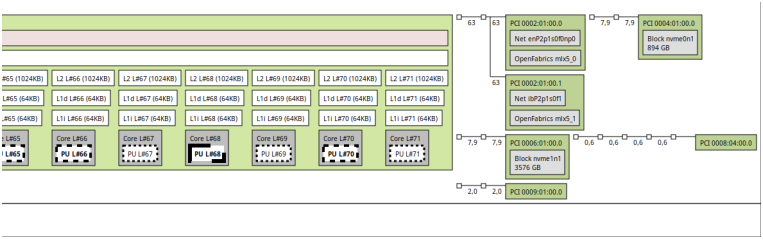
kraken : Intel Xeon x86-64 CPU + Nvidia Ampere A30

- **About kraken computing platform**
  - **Intel Xeon, x2 sockets, x16 cores**
  - **4 GPU are A30**



- on kraken : `lstopo-no-graphics -p lstopo_kraken.xml`
- on desktop : `lstopo -i lstopo_kraken.xml -l lstopo_kraken.png`

- **About calypso computing platform**
  - **Nvidia ARM Grace, 1 socket, x72 cores**
  - **1 GPU H100**



- on calypso-grace : `lstopo-no-graphics -p lstopo_calypso.xml`
- on desktop : `lstopo -i lstopo_calypso.xml -l lstopo_calypso.png`

Hands-On 00 : build an application on calypso

Minimal environment for building a GPU application on calypso :

```
module load gcc/12.3.0_arm tools/cmake/3.29.3_arm nvidia/cuda/12.4
```

Hands-On 00 : launch a job on `calypso`

Follow instructions from exercises/00-know-your-hardware/Readme.md
Purpose :

- Get familiar with `calypso` supercomputer environment
  - use `lstopo` / `lscpu` to explore compute platform features
  - modulefile environement to build a simple OpenMP/Cuda application
  - learn how to launch a job through Slurm
  - very basic introduction to performance measurements using `saxpy` example
  - Have an idea of the maximum hardware memory bandwidth on a GPU compute node

Evaluating Peak Memory bandwidth

How to determine the **peak** hardware memory bandwidth of your compute platform ?

- **Multicore CPU** (e.g. Intel Skylake) :
  - Memory type ? e.g. DDR4-2666
  - Number of channels ? e.g. 6
  - Max BW = # NbOfChannel $\times$ Frequency(GHz) $\times$ BusWidth/8 (Bytes) $\times$ # NbOfSockets
  - e.g. on TGCC/IRENE, BW $= 6 \times 2.6 \times 64/8 \times 2 = 256$ GBytes per node
- **Manycore CPU** (e.g. Intel KNL) :
  - depends on HBM configuration (CACHE, FLAT, HYBRID)
  - e.g. KNL on TGCC/IRENE configured in CACHE mode, BW $\geqslant$ 400 GBytes/s
- **NVIDIA GPU** (e.g. Pascal P100) :
  - Use sample application `deviceQuery` to retrieve hardware spec.
  - # Memory Clock rate : 715 Mhz
  - # Memory Bus Width : 4096-bit
  - $BW = 732.1$ Gbytes/s
- **NVIDIA GPU** (e.g. Pascal V100) :
  - $BW = 898.0$ Gbytes/s

Evaluating Peak Memory bandwidth

What about **achievable** memory bandwidth ?

- Use the stream benchmark. e.g. https://github.com/UoB-HPC/BabelStream
  - Copy : $C[i] = A[i]$
  - Trias : $A[i] = B[i] + scalar * C[i]$
- On **TGCC/Irene/Skylake** (2 sockets per node), one can measure :
  - copy : 190 GBytes/s (74 % of peak)
  - triad : 160 GBytes/s (63 % of peak)
- On **TGCC/Irene/KNL** (1 socket), one can measure :
  - copy : 260 GBytes/s (65 % of peak)
  - triad : 330 GBytes/s (80 % of peak)
- On **NVIDIA P100** :
  - copy : 530 GBytes/s (72 % of peak)
  - triad : 550 GBytes/s (75% of peak)
- On **NVIDIA V100** :
  - copy : 650 GBytes/s (72 % of peak)
  - triad : 860 GBytes/s (95% of peak)

revisit Stream Benchmark with kokkos, see
https://github.com/pkestene/kokkos-proj-tmpl/blob/master/src/KokkosStream.hpp
See also https://github.com/RRZE-HPC/TheBandwidthBenchmark