

Kokkos/C++ training

Performance portability

Pierre Kestener

December 11-13th, 2023, [Cerfacs](#), Toulouse



Funded by
the European Union

 CERFACS

The Cerfacs logo features a stylized blue Greek letter sigma (Σ) followed by the word "CERFACS" in a bold, black, sans-serif font.

 CCFR CENTRE
DE COMPÉTENCE
HPC.HPDA.IA

The CCFR logo consists of the letters "CCFR" in a large, gold-colored serif font. To the left is a stylized "C" formed by two overlapping semi-circles. To the right, the words "CENTRE DE COMPÉTENCE HPC.HPDA.IA" are written in smaller, gold-colored capital letters.



Monday, December 11th, 2023 : Kokkos tutorial

- ▶ **Introduction to performance portability**, hardware trends, programming models for heterogeneous computing platforms
- ▶ GPU Computing / Cuda refresher
- ▶ supercomputer kraken : a CPU (AMD Rome) + GPU (Nvidia Amepere A30) computing platform : short overview
- ▶ **Hands-on 00** : Get familiar with kraken 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 programming 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



Tuesday, December 12th, 2023 : Kokkos tutorial

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



Wednesday, December 13th, 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**
⇒ performance measurement for several Kokkos backends (OpenMP, CUDA)

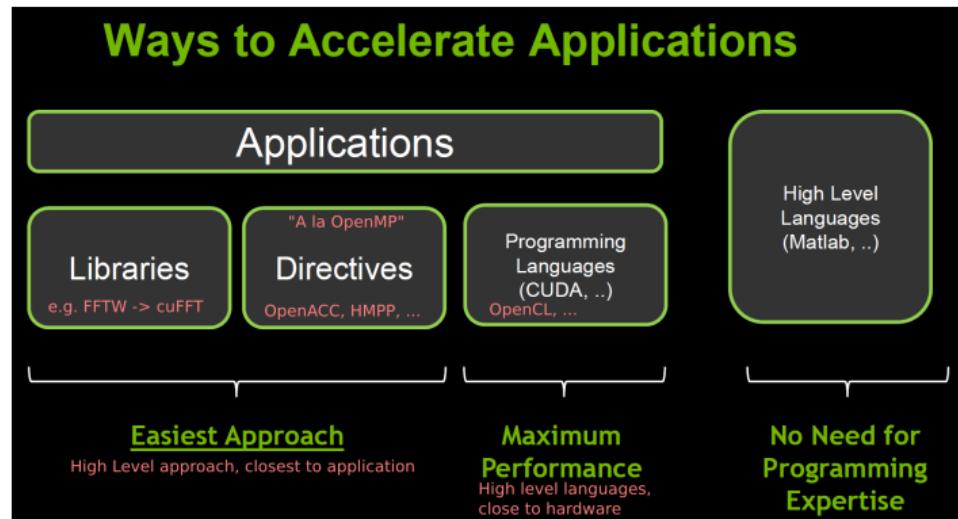
1. General Introduction



- ▶ 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 algorithmic patterns,
 - ▶ data containers.



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 is about...

- ▶ ... reaching exaflops = 10^{18} flop/s
 - ▶ 1.3 exaflops of aggregate peak flops
 - ▶ ~ 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**
 $P \in [20 - 40] \text{ MW}$
- ▶ 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)

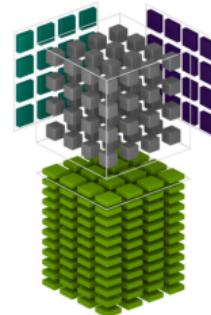
current #2 @top500



...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)





- ▶ **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-licensed 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 (\sim 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 Japan – 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



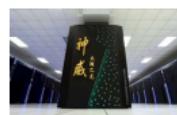
1.68 EF peak, 1.1 EF HPL (#1 top500)
9472 nodes, 100% hybrid, 21MW



>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>



EUROHPC, MAIN DIRECTIONS FOR #2 REGULATION (2021-2027)

2 main technological directions, 7B€ for the second phase



➤ Infrastructure deployment

	2019 & 2020	2021	2022	2023	2024	2025	2026	2027	#EuroHPC (high performance computing) Joint Undertaking
HPC Infrastructure	pre-exascale + petascale HPC systems		Several pre-exascale systems and exascale HPC systems			exascale and post-exascale HPC systems			1 st Exascale CFEI published by EuroHPC on dec 2021 → Selection of the German proposal (at FZJ) called JUPITER EVIDEN an atos business
Quantum Infrastructure	Quantum simulators interfacing with HPC systems		1 st generation of quantum computers + quantum simulators interfacing with HPC systems		2 nd generation of quantum computers + quantum simulators				→ 2 nd Exascale CFEI published on 14 December 2022

➤ Successive R&D calls for proposals including :

- an Exascale pilot – Consortium EUPEX (lead Atos)
- A quantum pilot – Consortium HPCQS

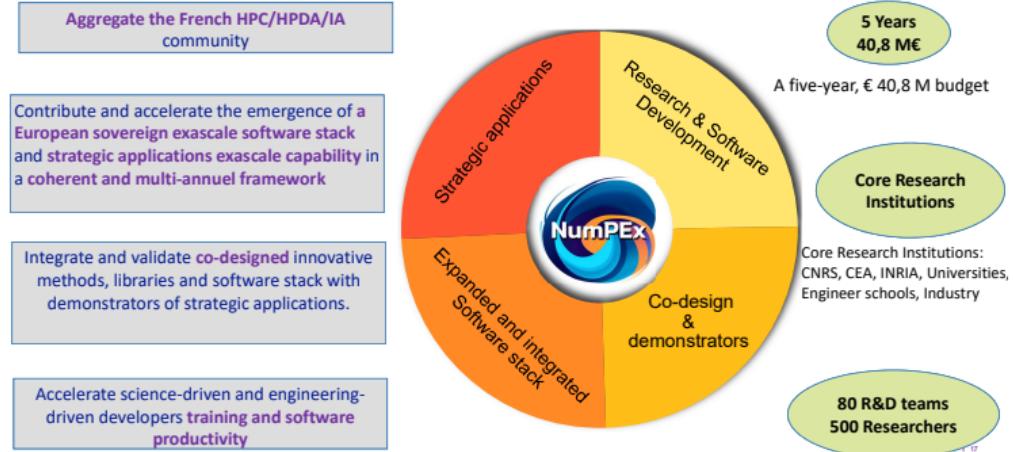


ORAP#51 | 17/10/2023 | 11

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



Toward Exascale applications and usages : The NumPEx project



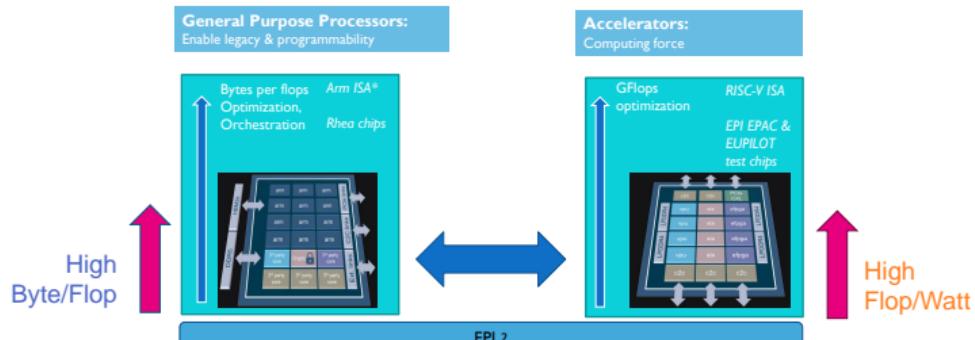
source : <http://orap.irisa.fr/wp-content/uploads/2023/10/01-GENCI-ORAP51.pdf>



—What?



EU chips fit for HPC usage - at Exascale level



Copyright © European Processor Initiative 2023

ORAP-Workshop 2023, 17 October

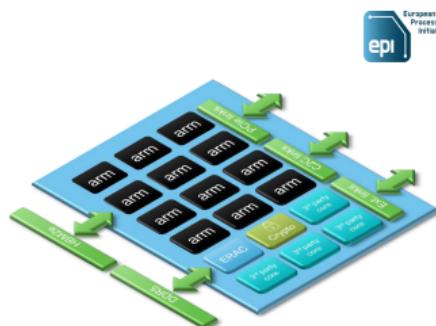
SIPERL 6

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



— Rhea Key Features

Element	Features
Core	<ul style="list-style-type: none"> - Arm ISA - Neoverse V1 cores - SVE 256 per core supporting 64/32/BF16 and Int8 - ArmVirtualization extensions
SoC	<ul style="list-style-type: none"> - NoC:Arm mesh fabric - Advanced RAS including Arm RAS extensions - Link protection for NoC & high-speed IO - ECC support for selected memory
Cache	<ul style="list-style-type: none"> - Large L3 (Shared Level Cache, SLC) - RAS supported for all cache levels
Memory	<ul style="list-style-type: none"> - HBM2e - And DDR5 - ECC for memory and link protection for controllers
High Speed I/O	<ul style="list-style-type: none"> - PCIe, CCIX & CXL - Root and endpoint support
Other I/O	<ul style="list-style-type: none"> - USB, GPIO, SPI, I2C
Power Management	<ul style="list-style-type: none"> - Power management block to optimize perf/Watt across use cases and workloads.
Security Block Support	<ul style="list-style-type: none"> - Secure boot and secure upgrade - Crypto - True random number generation



Design targets:

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

ORAP-Workshop 2023, 17 October

— SIPE/IRL 16

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

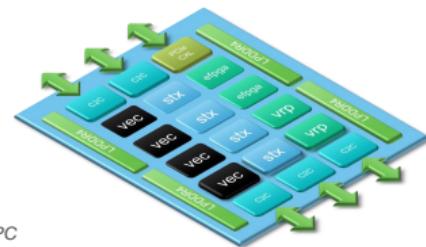


—EPAC Vision and Contributions



- **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



ORAP-Workshop 2023, 17 October

Copyright © European Processor Initiative 2023

SIPeRFL 22

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



— EPI2 timeline

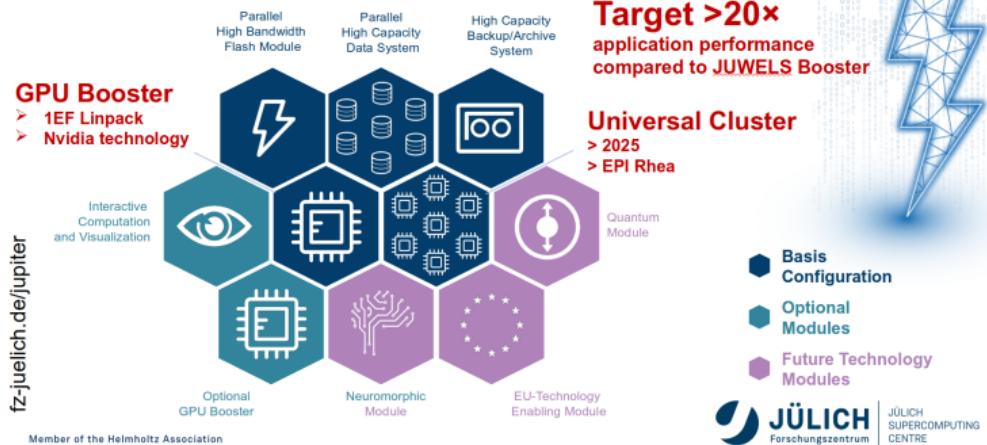


ORAP-Workshop 2023, 17 October

Copyright © European Processor Initiative 2023

 SIPeRl 30³⁰source : <http://orap.irisa.fr/wp-content/uploads/2023/10/06-SIPeRl-ORAP51.pdf>

JUPITER – MODULAR EXASCALE COMPUTER



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



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](#)



- ▶ 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 2¹ 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)
- ▶ emerging hybrid programming model : MPI + X
- ▶ What is X ? OpenMP, OpenAcc, ..., Kokkos, RAJA, ...
- ▶ Even at node level MPI+X is required : e.g. KNL

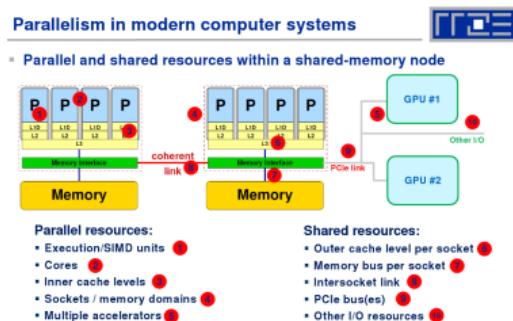
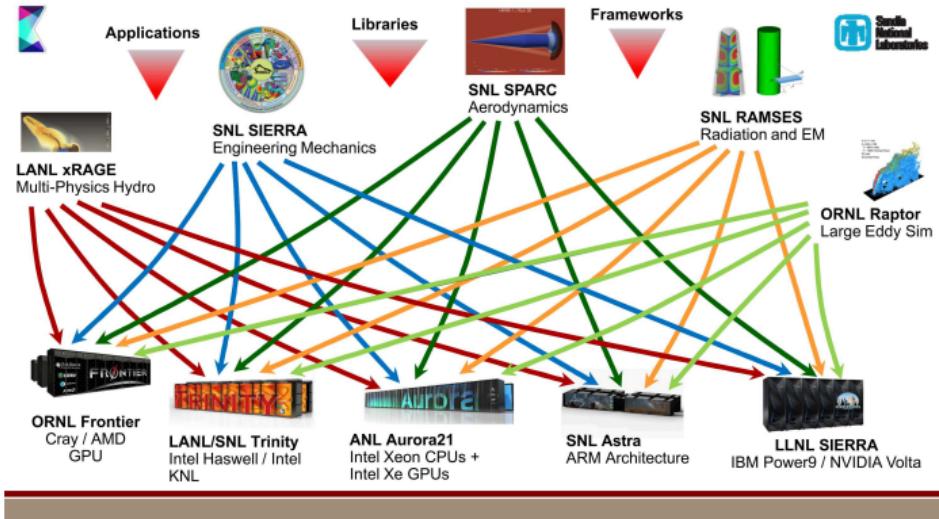


Figure 1 – Multi-core node summary, source : multicore tutorial (SC12) by G. Hager and G. Wellein

2. What is Performance Portability ?

cea

Why do we need performance portable programming models ?

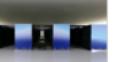


source : Jeff Miles, [Evolution of the Kokkos ecosystem](#), 2020

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 ^(e)

(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

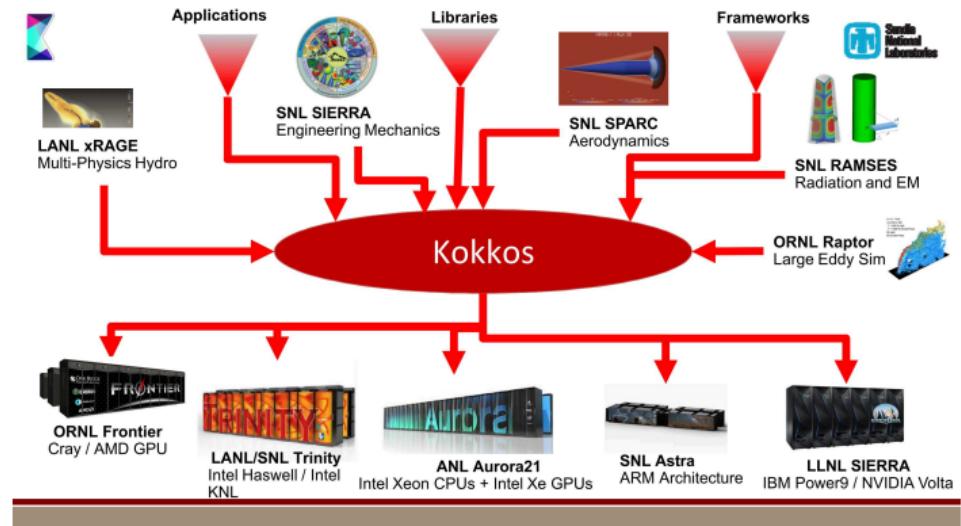


10 LOC / hour ~ 20k LOC / year

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



- ▶ 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 platforms²
- ▶ 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](#)



- ▶ 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 platforms²
- ▶ 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](#)



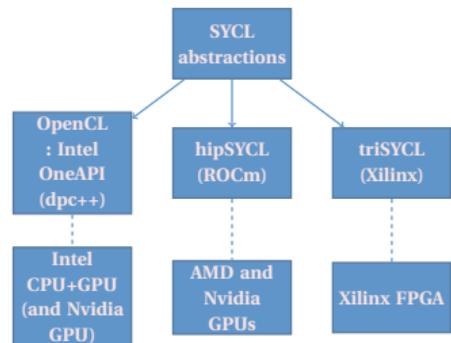
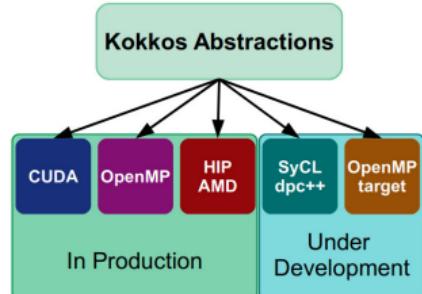
Developer productivity versus optimization

- ▶ 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 × 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

⇒ Constructs that can be used to express operations without going into details

- ▶ 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 ≥ 4.5 (nvhpc, Clang, GNU, ...)
 - ▶ OpenACC ≥ 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



additional features :

- **memory management**,
- **data containers**,



- ▶ 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 **maximum 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	✓	✓	✓	-	-	✓	✓	-

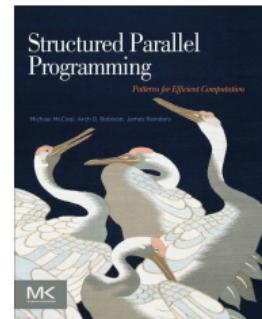
*Other names and brands may be claimed as the property of others.
© 2014 Intel Corporation.





- ▶ **pattern** : a basic structural entity of an algorithm

- ▶ book Structured Parallel Programming : Patterns for Efficient Computation



- ▶ implementation : Intel TBB, OpenMP, OpenACC and many others

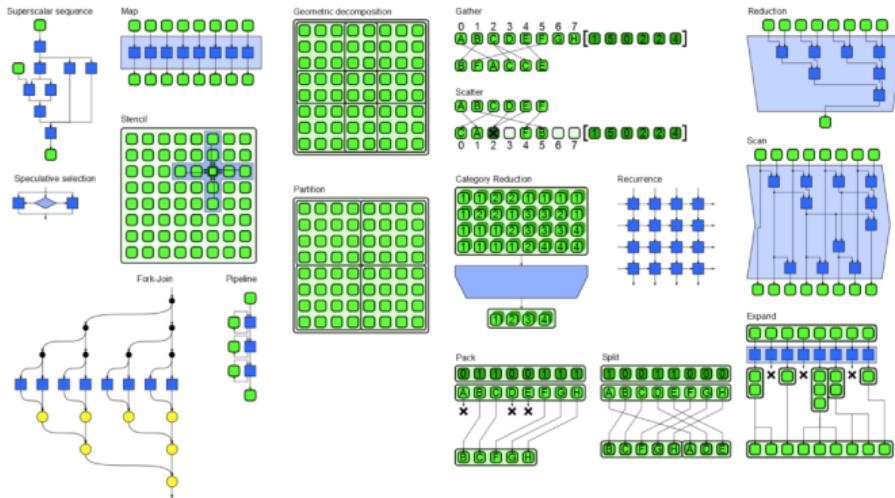
- ▶ OpenMP/OpenAcc for GPU/XeonPhi : pattern-based comparison : map, stencil, reduce, scan, fork-join, superscalar sequence, parallel update

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. Hebenstreit, J. Reinders, A. Robison, M. McCool



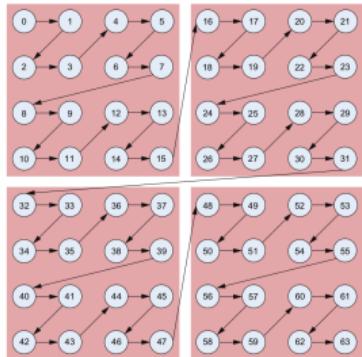
- ▶ **passive libraries** : a collection of subroutines
- ▶ **active libraires** : take an active role in compilation (specialize algorithms, tune themselves for target architecture).

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](#)



- ▶ 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	naive 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	2.09	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	naive 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	--



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

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

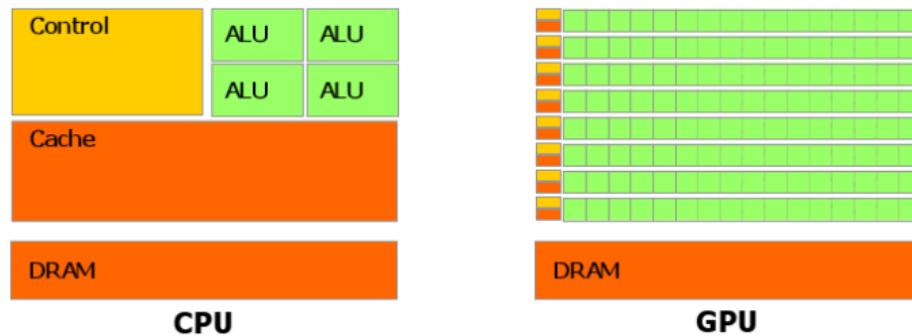


- ▶ Different goals produce different designs :
 - ▶ **CPU** must be good at everything, parallel or not
 - ▶ **GPU** assumes work load is highly parallel

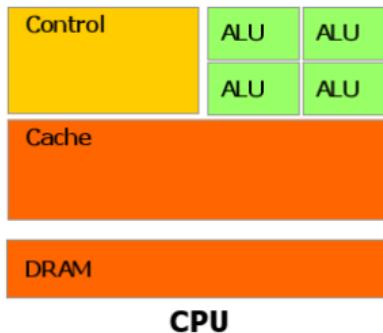
Architecture design differences between manycore GPUs and general purpose multicore CPU ?



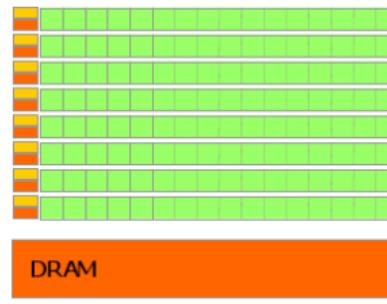
- ▶ **CPU** design goal : optimize architecture for sequential code performance : **minimize latency experienced by 1 thread**
 - ▶ **sophisticated** (i.e. large chip area) **control logic** for instruction-level parallelism (branch prediction, out-of-order instruction, etc...)
 - ▶ **CPU have large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

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

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

CPU



GPU

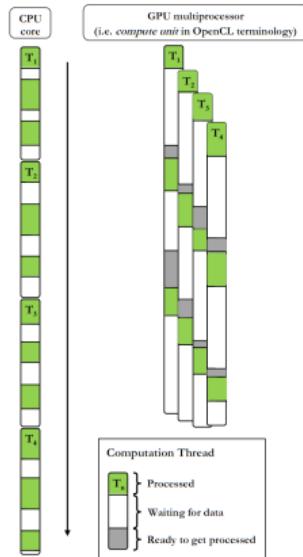
► **CPU (latency oriented architecture)**

- 1 thread \Leftrightarrow 1 core

► **GPU (throughput oriented architecture)**

- $\# \text{ threads} \gg \# \text{ cores}$

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



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



- ▶ **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...





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**



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

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

SMs # CUDA cores/SM # FP ops/cy

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



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	~19.5 TFlop/s
Stream BW (meas.)	2 x 180 GB/s	1400 GB/s
Transistors / TDP	~2x40 Billion / 2x225 W	54 Billion/400 W

Basic Node Architecture

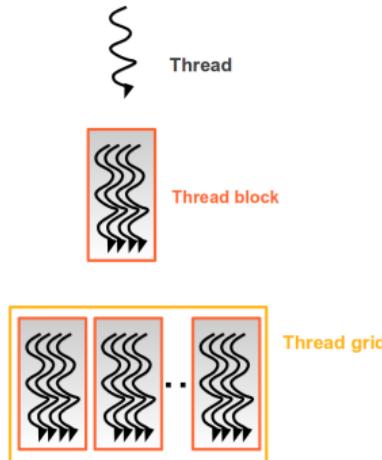
(c) NHR@FAU 2023

35

reference : [Node-level performance engineering](#), training, HLRS, June 2023

- ▶ 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

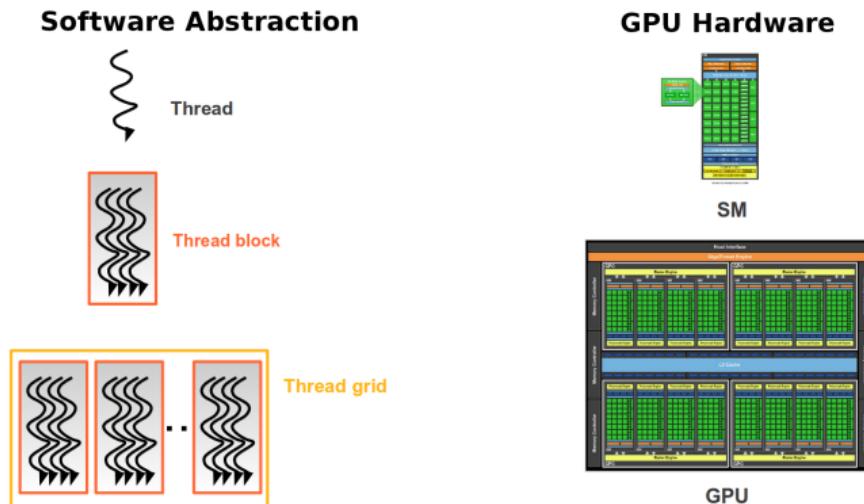
Software Abstraction



GPU Hardware

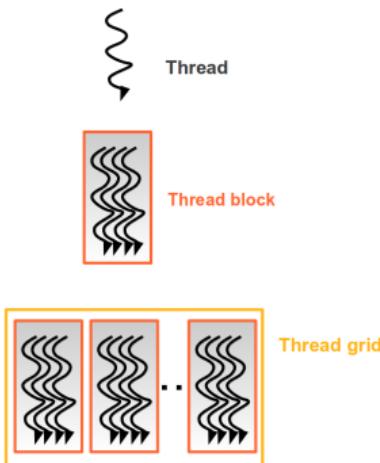


- ▶ Total number of threads must/need be quite larger than number of cores
- ▶ Thread block : logical array of threads, large number to hide latency
- ▶ Thread block size : control by program, specify at runtime, better be a multiple of warp size (i.e. 32)

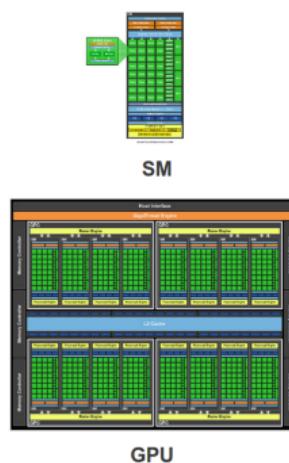


- ▶ Must give the GPU enough work to do ! : if not enough thread blocks, some SM will remain idle
- ▶ Thread grid : **logical array of thread blocks** distribute work among SM, several blocks / SM
- ▶ Thread grid : chosen by program at runtime, can be the total number of thread / thread block size or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

Software Abstraction

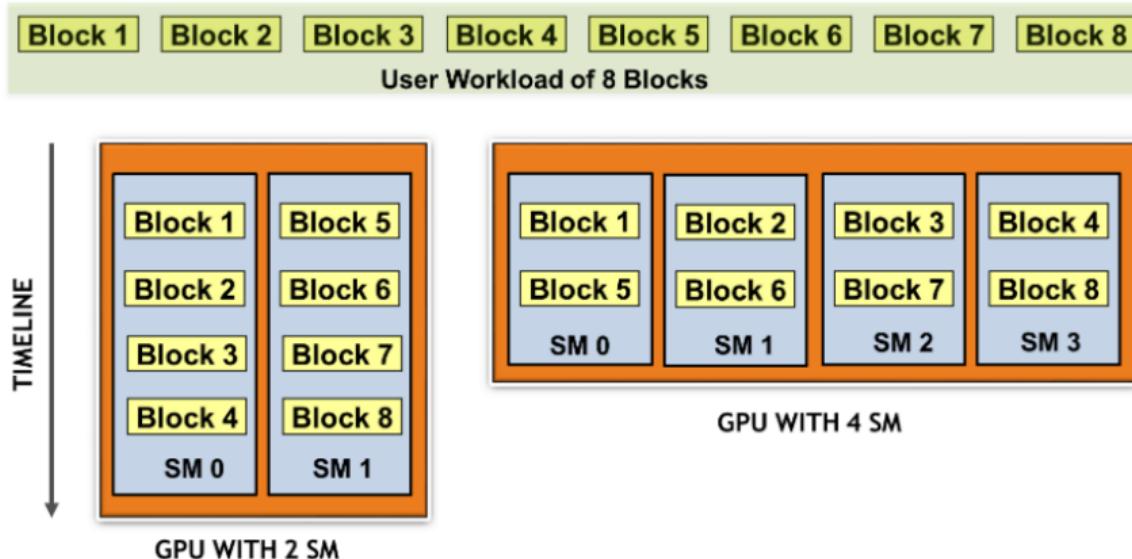


GPU Hardware



Why two levels of hierarchy block / grid ?

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



3. kraken computing platform



► About kraken computing platform

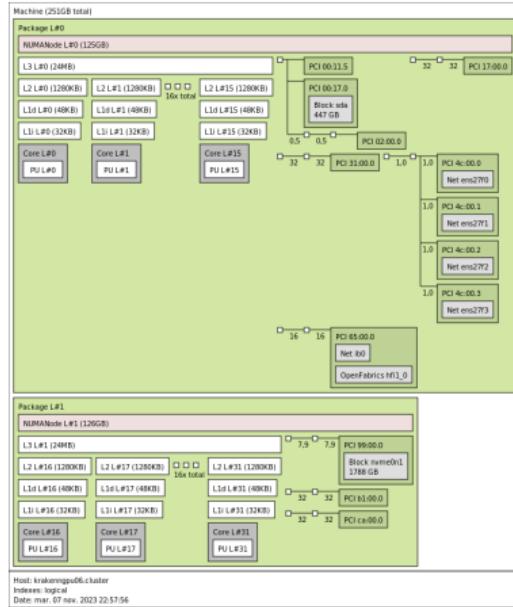
- ▶ Minimal information about software environment, how to build and run an application, submit a job on a GPU node of kraken
- ▶ Examples of job submission scripts :
[exercises/job.sh](#)
- ▶ How to run job ? sbatch ./job.sh

```
#!/bin/bash
#SBATCH -J compute_info
#SBATCH -N 1
#SBATCH --partition gpua30
#SBATCH --gres=gpu:a30:4
#SBATCH --ntasks-per-node=1
#SBATCH --mail-user=me@somewhere.org
#SBATCH --mail-type=END
cd $SLURM_SUBMIT_DIR

module load lib/hwloc/2.1.0
lstopo -p lstopo_kraken.xml
cat /proc/driver/nvidia/version
nvidia-smi
lscpu
```

► About kraken computing platform

- Intel Xeon, x2 sockets, x16 cores
- 4 GPU are A30





Follow instructions from [exercises/00-know-your-hardware/Readme.md](#)

Purpose :

- ▶ Get familiar with kraken supercomputer environment
 - ▶ use lstopo / lscpu to explore compute platform features
 - ▶ modulefile environment 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



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



What about **achievable** memory bandwidth ?

- ▶ Use the stream benchmark. e.g. <https://github.com/UoB-HPC/BabelStream>
 - ▶ Copy : $C[i] = A[i]$
 - ▶ Trias : $A[i] = B[i] + \text{scalar} * C[i]$
- ▶ 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>