Round 3: Existing ecosystems, design methods and challenges.

Let’s discuss ?

What this presentation is or is not ?

Outline

Existing ecosystems

All in one solutions and others of interest.

Design methods and challenges

Language and libraries opportunities & challenges

Discussion

Let’s discuss about what’s interesting or not, difficult or not...

Existing ecosystems : All in one solution.

Kokkos

*“**Kokkos** Core implements **a programming model in C++** for writing **performance portable applications** targeting **all major HPC platforms**. For that purpose it provides **abstractions for both parallel execution of code and data management**. Kokkos is designed to target complex node architectures with N-level memory hierarchies and multiple types of execution resources.”*

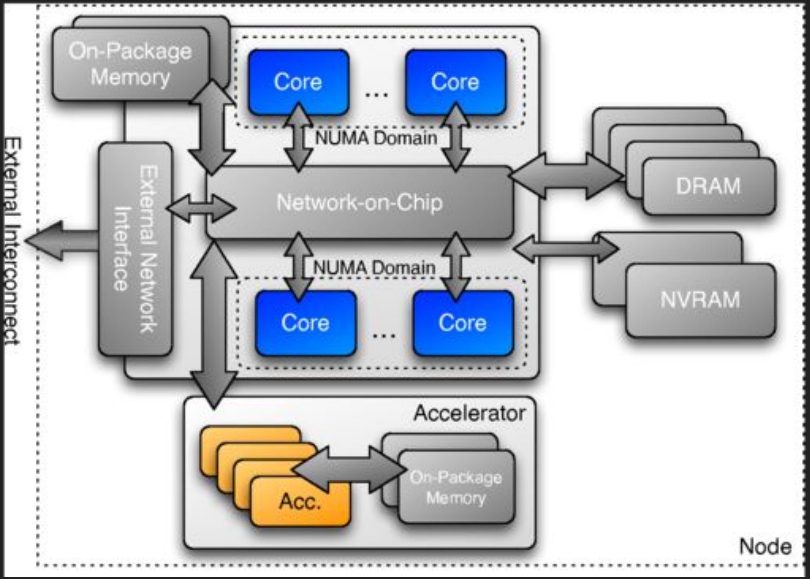
Kokkos Spaces

Execution spaces

Bind parallel work to the instantiation of an execution space.

Multicores + 1 GPU = 2 execution spaces

Compiling code and the dispatching it to different instances is abstracted by the Kokkos model.



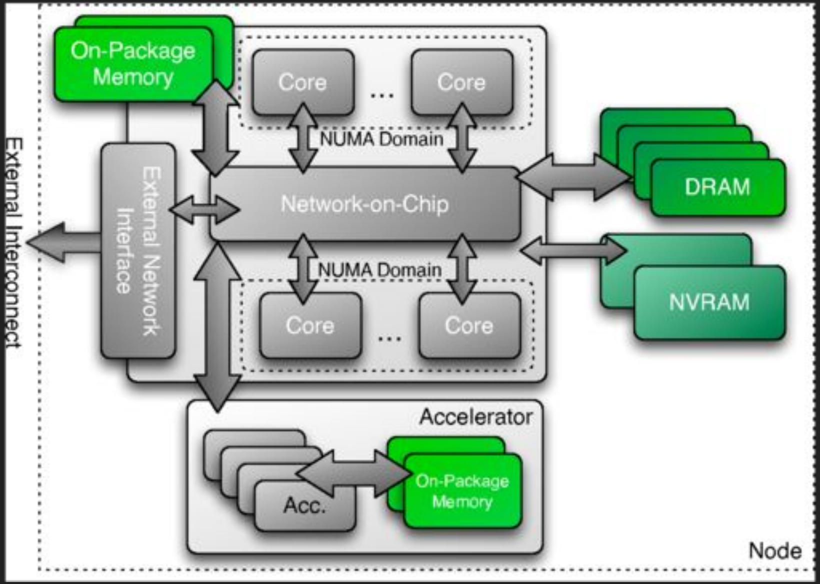
Source : Kokkos

Kokkos Spaces

Memory spaces

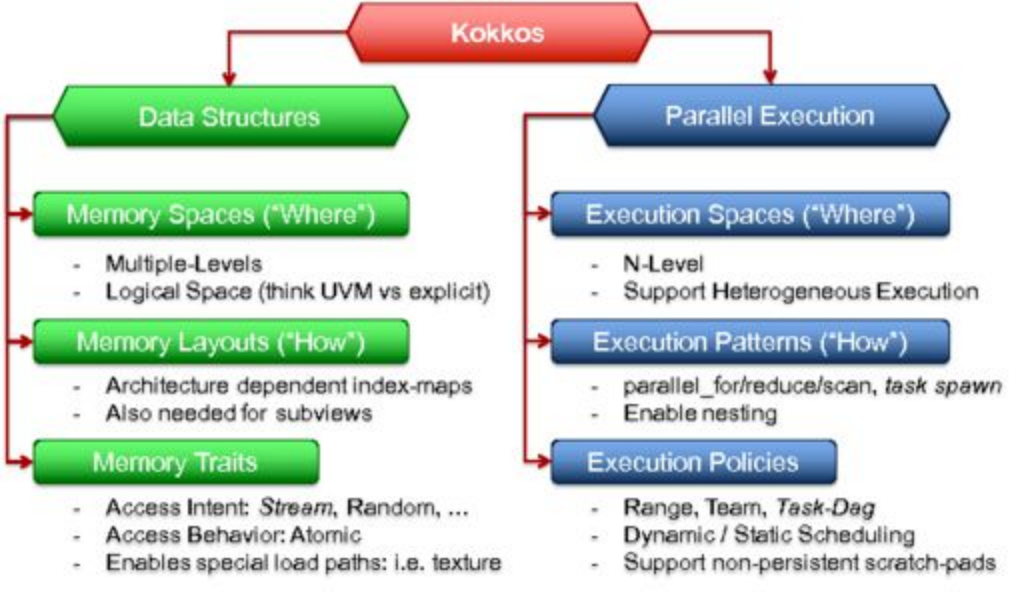
The programmer can requests data storage allocations through instances of specific memory spaces.

A multicore processor may have multiple memory spaces available.



Source : Kokkos

Kokkos Programming Model



Kokkos

Data Structures

Memory Spaces ("Where")

- Multiple-Levels
- Logical Space (think UVM vs explicit)

Memory Layouts ("How")

- Architecture dependent index-maps
- Also needed for subviews

Memory Traits

- Access Intent: *Stream*, Random, …
- Access Behavior: Atomic
- Enables special load paths: i.e. texture

Parallel Execution

Execution Spaces ("Where")

- N-Level
- Support Heterogeneous Execution

Execution Patterns ("How")

- parallel_for/reduce/scan, *task spawn*
- Enable nesting

Execution Policies

- Range, Team, *Task-Dag*
- Dynamic / Static Scheduling
- Support non-persistent scratch-pads

Source : Kokkos

Kokkos

High level of abstraction and portability

- Rich backends options (CUDA, HIP, SYCL, HPX, OpenMP and C++ threads)

Standard enthusiast

- std::mdspan & std::linalg reference implementations

Community / documentation

Available through Spack package manager.

Parallelism expressivity tied to few patterns mainly loop based.

- Task paradigm support
- Asynchrony

Composability outside of Kokkos ecosystem ?

Raja

*“**RAJA** is a software library of **C++ abstractions**, developed at Lawrence Livermore National Laboratory (LLNL), that **enable architecture and programming model portability for high performance computing** (HPC) applications.”*

“Mac and Windows laptops, parallel clusters of multicore commodity processors, and large-scale supercomputers with advanced heterogeneous node architectures that combine cutting edge CPU and accelerator (e.g., GPU) processors. Exposing fine-grained parallelism in a portable, high performance manner on varied and potentially disruptive architectures presents significant challenges to developers of large-scale HPC applications. [...] RAJA is one C++ abstraction layer that helps address this performance portability challenge.”

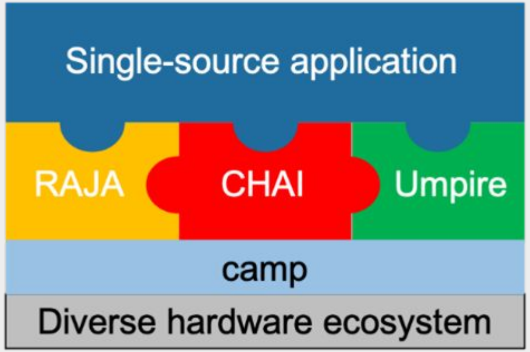
Raja

RAJA: C++ kernel execution abstractions

- Enables apps to target various programming model back-ends while maintaining **single-source** app code

CHAI: C++ array abstractions

- Automates data copies, giving look and feel of unified memory



Single-source application

RAJA

CHAI

Umpire

camp

Diverse hardware ecosystem

https://github.com/LLNL/RAJA

https://github.com/LLNL/CHAI

https://github.com/LLNL/Umpire

https://github.com/LLNL/camp

Umpire: memory API

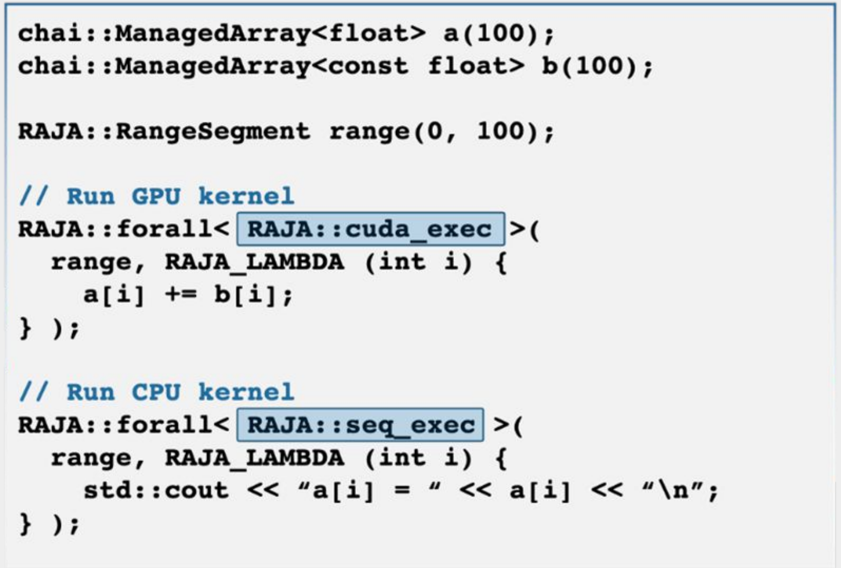
- Provides high performance memory operations, such as pool allocations. **Native C++, C, Fortran APIs**

camp: low-level C++ metaprogramming facilities

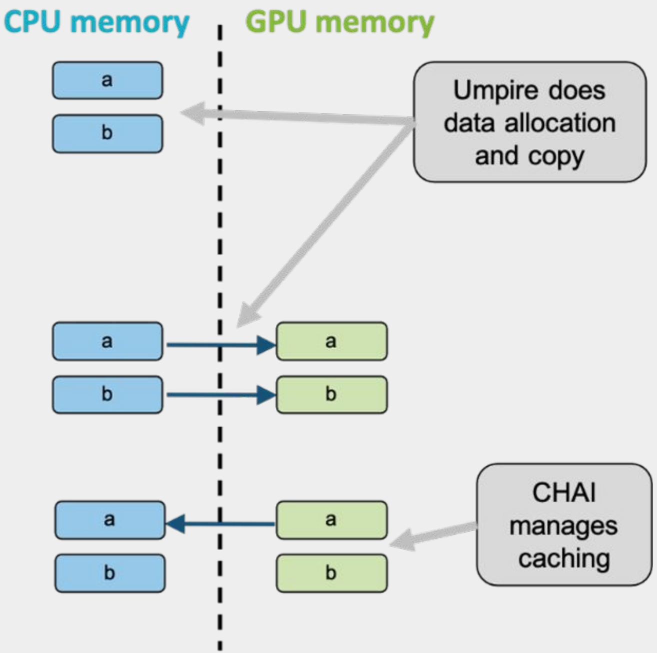
- Focuses on HPC compiler compatibility

Source : computing.llnl.gov

Raja

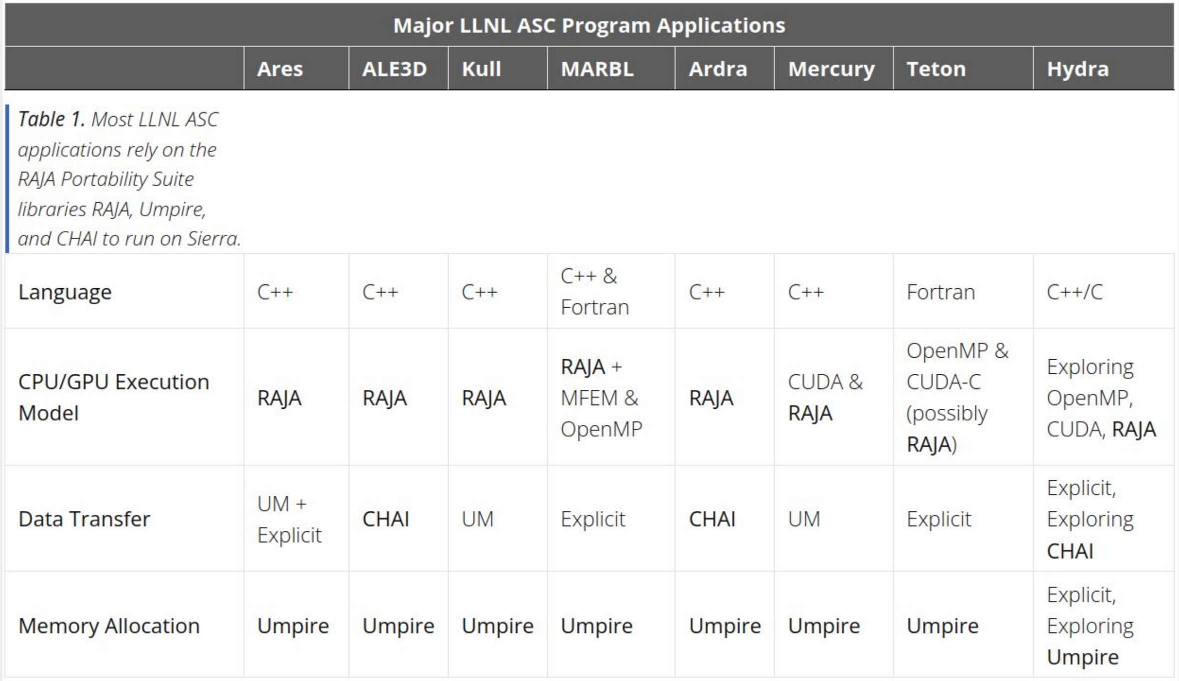


```
chai::ManagedArray<float> a(100);
chai::ManagedArray<const float> b(100);

RAJA::RangeSegment range(0, 100);

// Run GPU kernel
RAJA::forall< RAJA::cuda_exec >(
  range, RAJA_LAMBDA (int i) {
    a[i] += b[i];
} );

// Run CPU kernel
RAJA::forall< RAJA::seq_exec >(
  range, RAJA_LAMBDA (int i) {
    std::cout << "a[i] = " << a[i] << "\n";
} );
```



Source : computing.llnl.gov

Raja



Major LLNL ASC Program Applications								
	Ares	ALE3D	Kull	MARBL	Ardra	Mercury	Teton	Hydra
Table 1. Most LLNL ASC applications rely on the RAJA Portability Suite libraries RAJA, Umpire, and CHAI to run on Sierra.								
Language	C++	C++	C++	C++ & Fortran	C++	C++	Fortran	C++/C
CPU/GPU Execution Model	RAJA	RAJA	RAJA	RAJA + MFEM & OpenMP	RAJA	CUDA & RAJA	OpenMP & CUDA-C (possibly RAJA)	Exploring OpenMP, CUDA, RAJA
Data Transfer	UM + Explicit	CHAI	UM	Explicit	CHAI	UM	Explicit	Explicit, Exploring CHAI
Memory Allocation	Umpire	Umpire	Umpire	Umpire	Umpire	Umpire	Umpire	Explicit, Exploring Umpire

Source : computing.llnl.gov

Raja

Abstraction and portability

Modularity in the components

Popular

Standard compliance

Complex and aging design

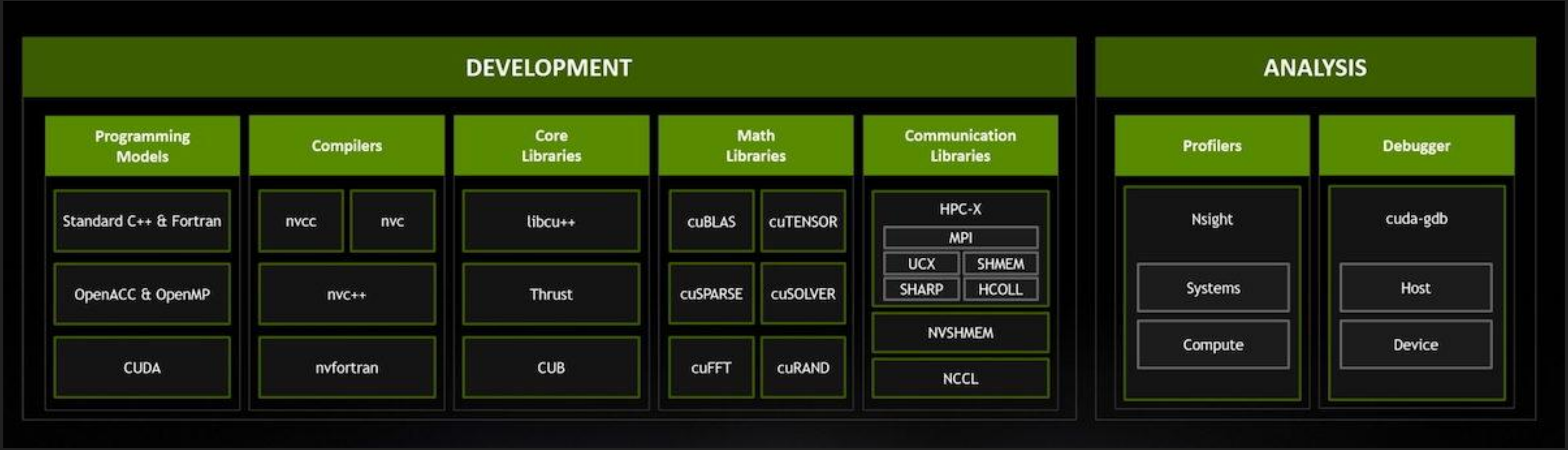
Parallelism expressivity tied to few patterns mainly loop based

- Task paradigm support
- Asynchrony

Composability outside of Raja ecosystem ?

Existing ecosystems : Vendors

NVIDIA



DEVELOPMENT

Programming Models
- Standard C++ & Fortran
- OpenACC & OpenMP
- CUDA

Compilers
- nvcc
- nvc
- nvc++
- nvfortran

Core Libraries
- libcu++
- Thrust
- CUB

Math Libraries
- cuBLAS
- cuTENSOR
- cuSPARSE
- cuSOLVER
- cuFFT
- cuRAND

Communication Libraries
- HPC-X
- MPI
- UCX
- SHMEM
- SHARP
- HCOLL
- NVSHMEM
- NCCL

ANALYSIS

Profilers
- Nsight
- Systems
- Compute

Debugger
- cuda-gdb
- Host
- Device

Source : NVIDIA

NVIDIA

High level to low level abstraction

Really standard enthusiast

- Unlocked composability

Some even works on AMD ?

std::execution reference implementation

Recent work shows a unification of the C++ ecosystem (thrust, libcuxx, CUB)

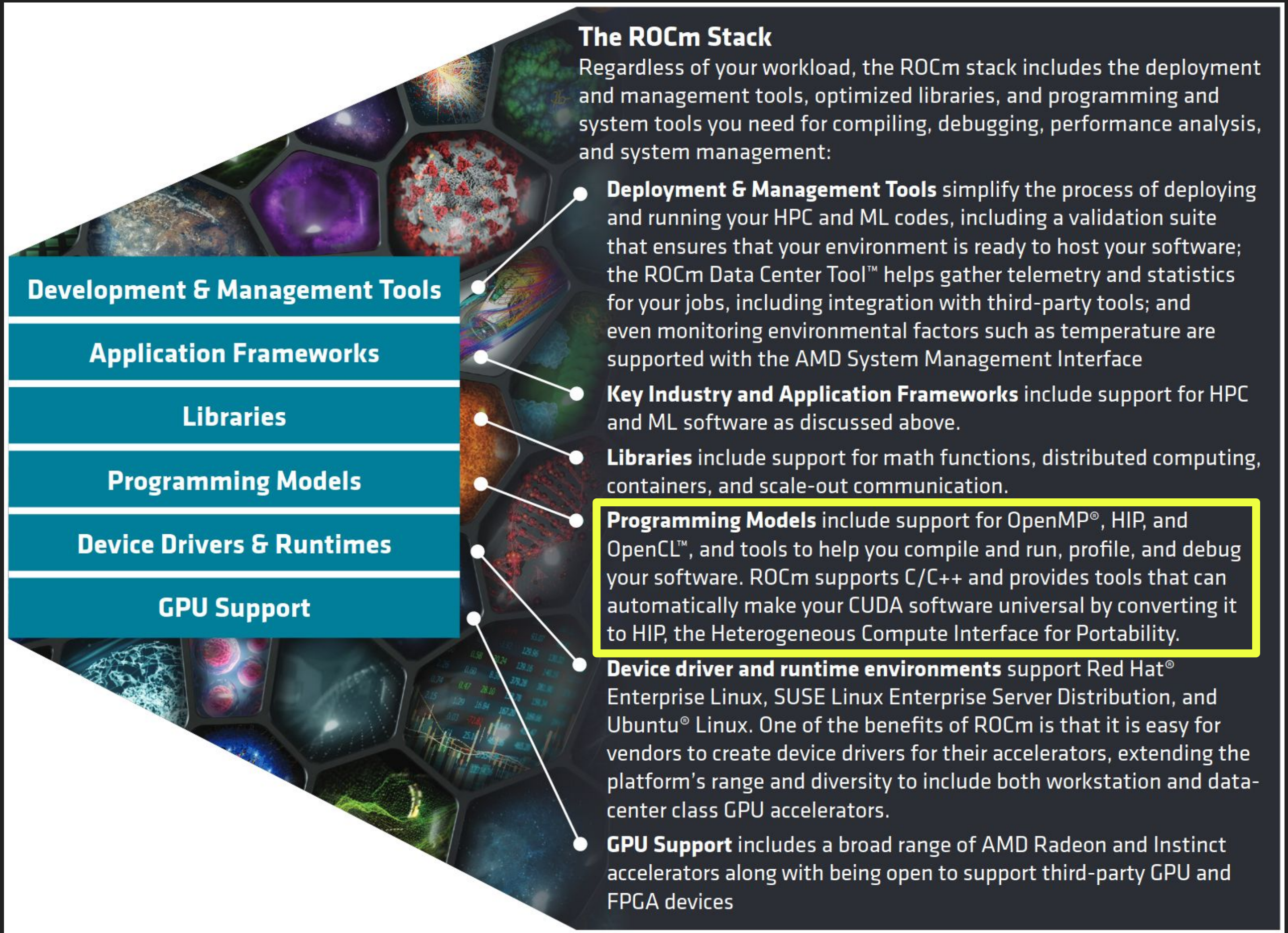
Documentation

A lot of components, not a clear and straightforward ecosystem

Containers

Tedious to deploy

AMD



Development & Management Tools

Application Frameworks

Libraries

Programming Models

Device Drivers & Runtimes

GPU Support

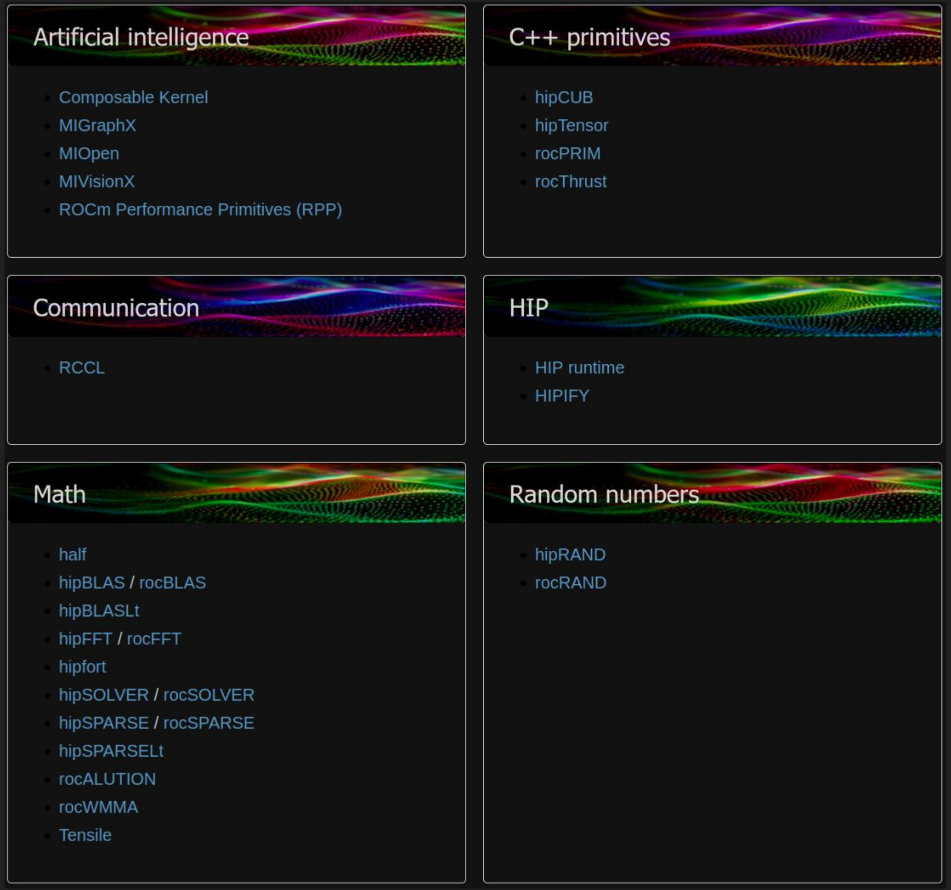
The ROCm Stack

Regardless of your workload, the ROCm stack includes the deployment and management tools, optimized libraries, and programming and system tools you need for compiling, debugging, performance analysis, and system management:

- **Deployment & Management Tools** simplify the process of deploying and running your HPC and ML codes, including a validation suite that ensures that your environment is ready to host your software; the ROCm Data Center Tool™ helps gather telemetry and statistics for your jobs, including integration with third-party tools; and even monitoring environmental factors such as temperature are supported with the AMD System Management Interface
- **Key Industry and Application Frameworks** include support for HPC and ML software as discussed above.
- **Libraries** include support for math functions, distributed computing, containers, and scale-out communication.
- **Programming Models** include support for OpenMP®, HIP, and OpenCL™, and tools to help you compile and run, profile, and debug your software. ROCm supports C/C++ and provides tools that can automatically make your CUDA software universal by converting it to HIP, the Heterogeneous Compute Interface for Portability.
- **Device driver and runtime environments** support Red Hat® Enterprise Linux, SUSE Linux Enterprise Server Distribution, and Ubuntu® Linux. One of the benefits of ROCm is that it is easy for vendors to create device drivers for their accelerators, extending the platform's range and diversity to include both workstation and data-center class GPU accelerators.
- **GPU Support** includes a broad range of AMD Radeon and Instinct accelerators along with being open to support third-party GPU and FPGA devices

Source : AMD

AMD



Artificial intelligence

- Composable Kernel
- MIGraphX
- MIOpen
- MIVisionX
- ROCm Performance Primitives (RPP)

C++ primitives

- hipCUB
- hipTensor
- rocPRIM
- rocThrust

Communication

- RCCL

HIP

- HIP runtime
- HIPIFY

Math

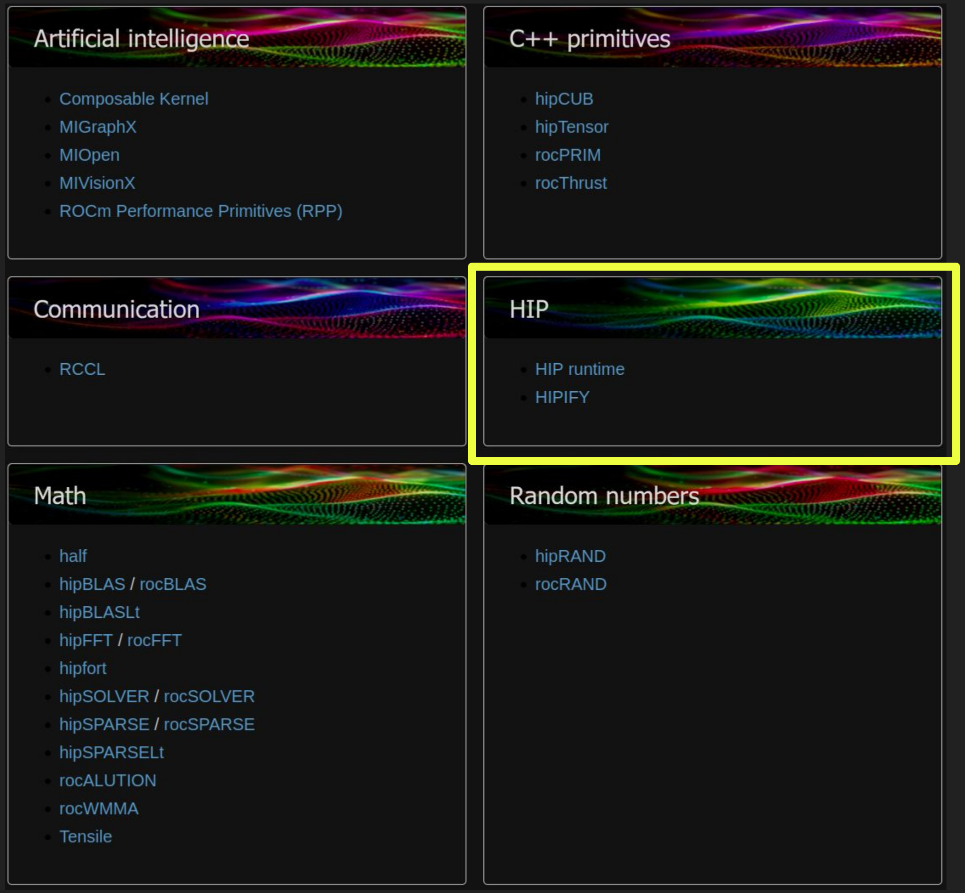
- half
- hipBLAS / rocBLAS
- hipBLASLt
- hipFFT / rocFFT
- hipfort
- hipSOLVER / rocSOLVER
- hipSPARSE / rocSPARSE
- hipSPARSELt
- rocALUTION
- rocWMMA
- Tensile

Random numbers

- hipRAND
- rocRAND

Source : AMD

AMD



Artificial intelligence

- Composable Kernel
- MIGraphX
- MIOpen
- MIVisionX
- ROCm Performance Primitives (RPP)

C++ primitives

- hipCUB
- hipTensor
- rocPRIM
- rocThrust

Communication

- RCCL

HIP

- HIP runtime
- HIPIFY

Math

- half
- hipBLAS / rocBLAS
- hipBLASLt
- hipFFT / rocFFT
- hipfort
- hipSOLVER / rocSOLVER
- hipSPARSE / rocSPARSE
- hipSPARSELt
- rocALUTION
- rocWMMA
- Tensile

Random numbers

- hipRAND
- rocRAND

Source : AMD

AMD ROCm

Follow the green rabbit, Neo.

Ok... HIP exists.

Follow the green rabbit, Neo.

Is following the green rabbit the path you want to take Neo ?

Intel

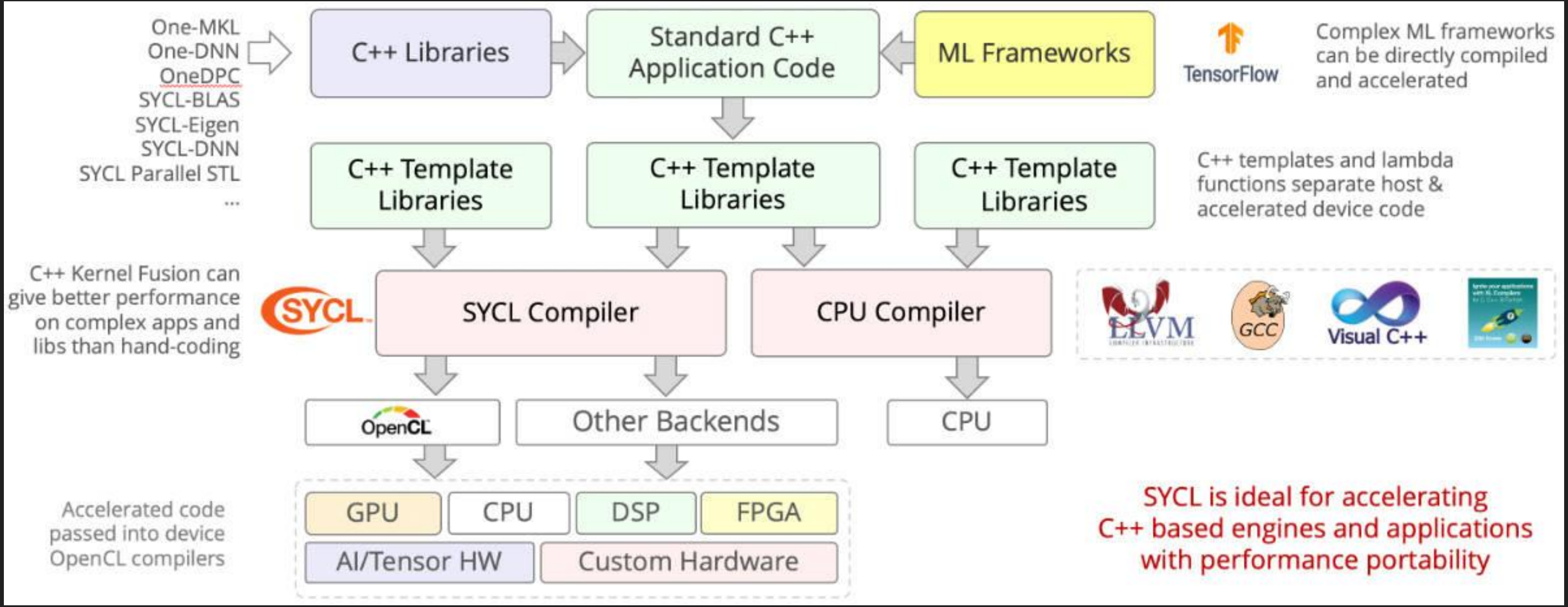
OneAPI

Mainly based on SYCL open standard

SYCL

“SYCL is an open industry standard for programming a heterogeneous system. The design of SYCL allows standard C++ source code to be written such that it can run on either an heterogeneous device or on the host.”

Intel



One-MKL
One-DNN
OneDPC
SYCL-BLAS
SYCL-Eigen
SYCL-DNN
SYCL Parallel STL
...

Complex ML frameworks can be directly compiled and accelerated

C++ templates and lambda functions separate host & accelerated device code

C++ Kernel Fusion can give better performance on complex apps and libs than hand-coding

Accelerated code passed into device OpenCL compilers

SYCL is ideal for accelerating C++ based engines and applications with performance portability

Source : Kronos

Intel

SYVL is an open standard

Compiler approach

Allows to target fancy architectures like FPGAs

AdaptiveCpp may be of interest

Specific compiler needed with complex architecture

OpenCL legacy

Support in the future ?

Needs backend support from vendors (they may have other plans)

Is it really composable ?

General design trends

Memory model

Unified Memory Space

Obliviate distributed computing challenges

Obliviate memory handling from the programming model

Yes but

Vendors tend to give access to both unified and non unified in their frameworks.

Execution model

Regular patterns to the rescue

Parallel loops and scans abstraction

Few propose task based approach with support for coarse to fine grain parallelism.

Some efforts are made to break barriers

and allow more asynchronism.

Part of this work is made to push further the standardisation (std::execution).

Programming model

Abstractions are constructed from recurrent patterns

- The algorithm. (the good)
- The data. (the bad)
- The machine. (the ugly)

Most of them put some makeup on the ugly through the memory model and the execution model.

- Mainly because the algorithms drives everything.

But we know that it is not true.

- Machines forces us to write application a certain way.
- Data rearrangements or locality can (and will) enhanced the algorithms so we need some latitude in software abstractions.

Programming model

Most of these ecosystems are or at least try to be standard compliant

With significant effort to contribute to the standard.

All are engaging to multidimensional abstractions

Through lightweight multidimensional views or multidimensional arrays/buffers (with different memory handling strategies)

std::mdspan ?

Views are a recurrent concept

It allows powerful memory abstraction with user specific data.

Programming model

Asynchronism ?

Vendors support and runtimes tends to go this way.

Kokkos is extending the task paradigm.

std::execution pushed by NVIDIA

But is std::execution on par with most advanced runtimes ?

No, but it’s a start to asynchronism and task support within a standard.

Distribution

Package managers not do frequent

Spack

Mainly based on CMake build system

C++ software stack deployment ?

Existing ecosystems : Others of interest & Community

Github Top 5 #cpp #hpc

arrayfire

General-purpose tensor library CPU-GPU

boost compute

GPU/parallel-computing library for C++ based on OpenCL.

gunrock

CUDA library for graph-processing designed specifically for the GPU.

eve

C++20 and onward implementation of a type based wrapper around SIMD extensions sets

nvidia cccl

CUDA C++ Core Libraries unifies three essential CUDA C++ libraries : Thrust, CUB & libcuxx

Others

A lot actually...

https://github.com/trevor-vincent/awesome-high-performance-computing?tab=readme-ov-file#software

Community

Some ressources :

https://notes.inria.fr/s/F8koaNZUF#

Reddit

https://www.reddit.com/r/cpp/

Slack / The cpp alliance

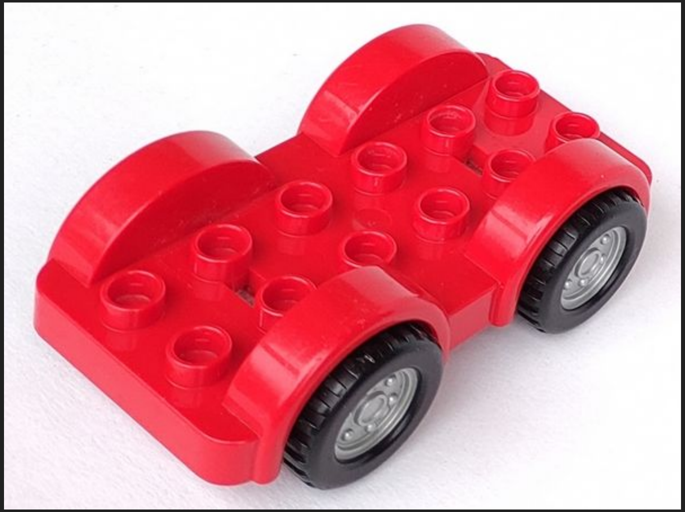
https://cppalliance.org/

Challenges

The Ferrari of Hpc ?

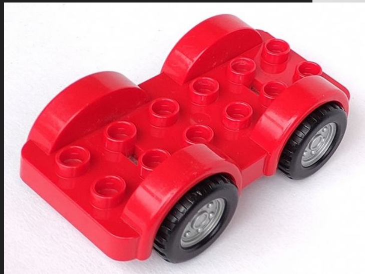
The Ferrari of Hpc ?

The Ferrari of Hpc ?

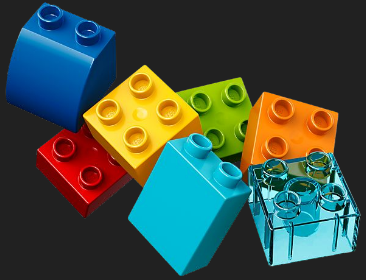


The Ferrari of Hpc ?

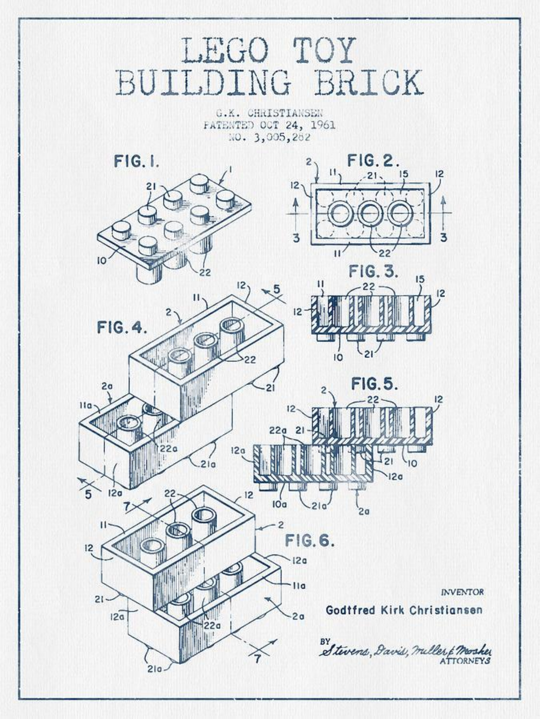
The Ferrari of Hpc ?



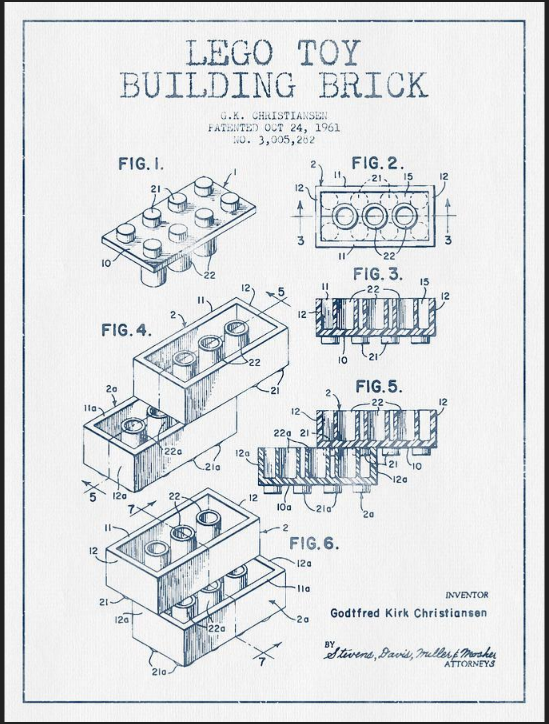
The Ferrari of Hpc ?

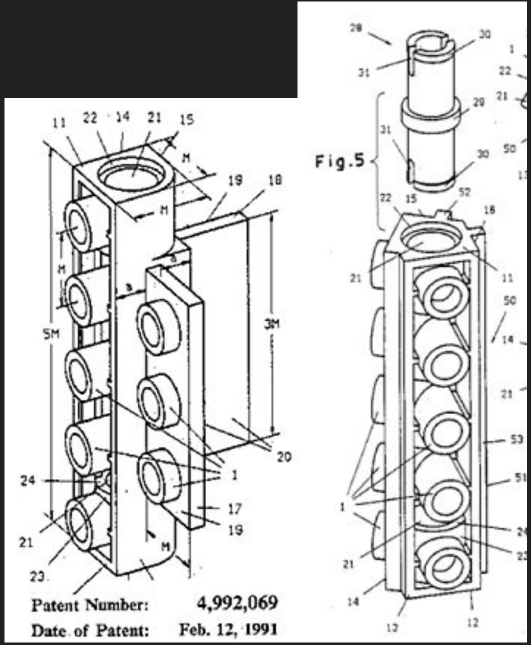


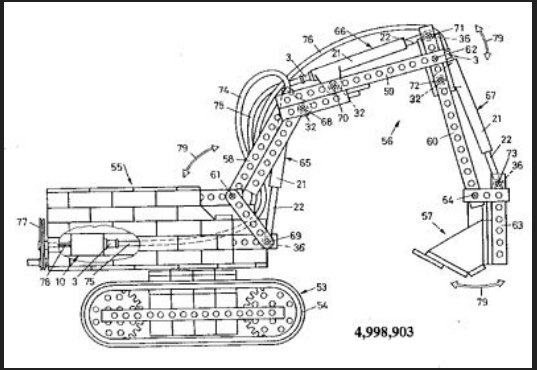
The Ferrari of Hpc ?



The Ferrari of Hpc ?







Let’s wrap it up & discuss !

C++ ecosystem ?

The needs ?

The direction ?