Introduction to performance portability. Mini-overview of Kokkos/C++ library.

Pierre Kestener[1]

[1] CEA Saclay, DRF, Maison de la Simulation

PATC, April 1st - April 2nd, 2018

location: IDRIS, training room

Monday, April 1st, 2019: Kokkos tutorial

- GPU Computing / Cuda refresh
- **Introduction performance portability**
- `ouessant` : a CPU (IBM Power8) + GPU (Nvidia Pascal P100) computing platform : short overview
- **C++ Kokkos: features overview**
- Hands-on 0: **retrieve Kokkos sources**, how to build, how to run a *helloworld* application, explore different configurations
- Hands-on 1: cross-checking **Kokkos + hwloc** is OK
- **Replay some tutorial slides from SC2016 for deeper Kokkos concepts**
- Hands-On 2: Simple example **SAXPY**
  ⇒ simplest computing kernel in Kokkos
- Hands-On 3: Simple example **Mandelbrot set**
  ⇒ 1D Kokkos::View + linearized index (+ asynchronous execution)
- **a Kokkos miniapp skeleton project with cmake**

Tuesday, April 2nd, 2019: Kokkos tutorial

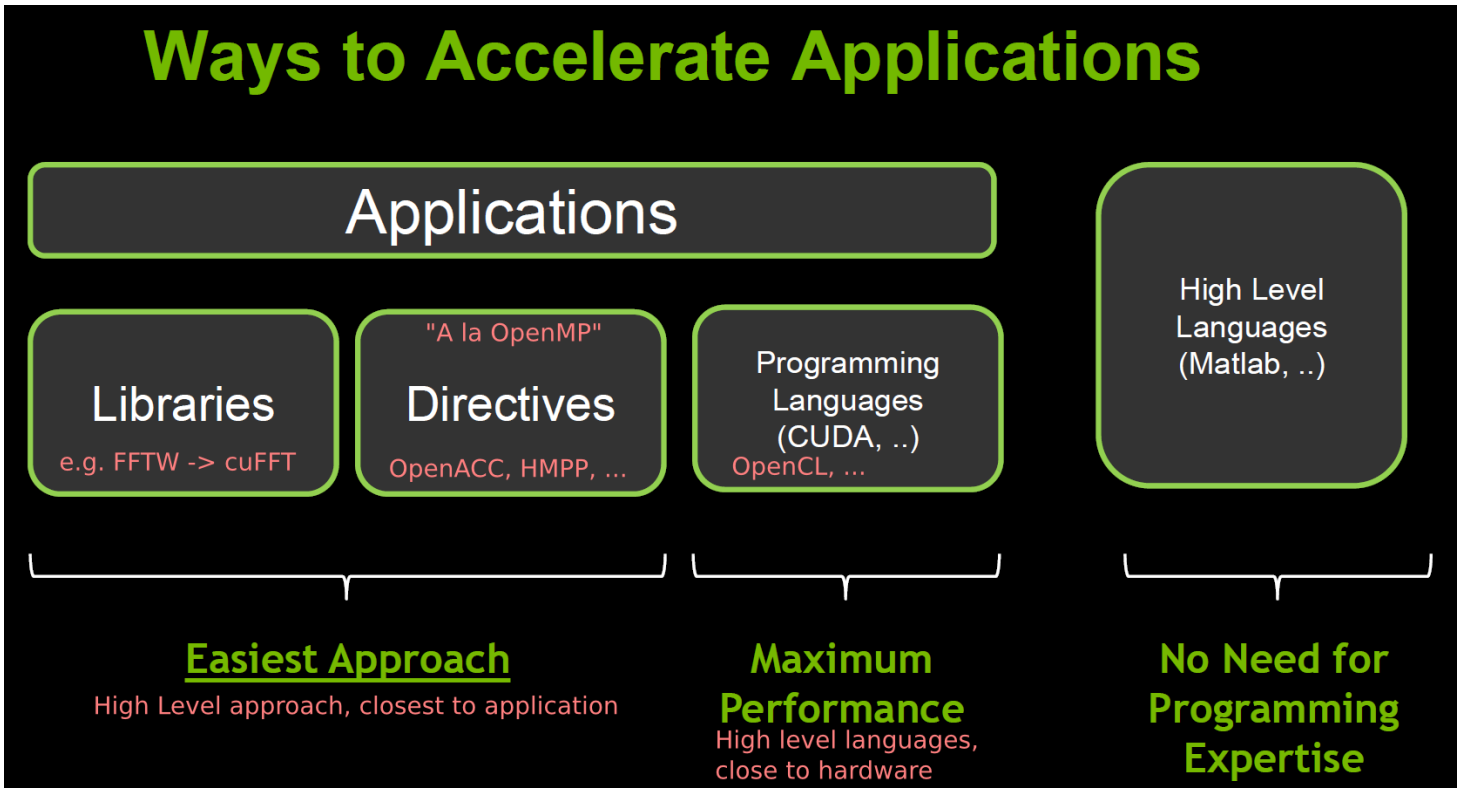
- Hands-On 4: Simple examples **Stencil + Finite Difference**
  ⇒ 2D Kokkos::View
- Hands-On 5: **Laplace exercice**
  ⇒ pure Kokkos versus Kokkos + MPI + hwloc (multiGPU)
- Hands-On 6: **Illustrate how to use random number generator in kokkos**
  ⇒ RNG 101, parallel compute π with Monte Carlo
- Hands-On 7: CSCS miniApp: **Fisher equation solver**
  ⇒ use Kokkos lambda
- Hands-On 8: CFD miniApp: **Euler solver**
  ⇒ performance measurement for several Kokkos backends (OpenMP, CUDA)

The pedagogical material is located in:
`/pwrwork/workshops/2019-04_patc` **on** `ouessant`

Content

- **Main HPC architectures and trends** multicore, manycore, GPU, FPGA, Power8/9, NVLink, ...
- **What is performance portability ?**
- **A good software abstraction / programing model(s) (?)**
  - **library, framework, programming models ?**
  - **Parallel programming patterns**
  - **Native language, directives, DSL ?**
- **As an example: a short overview of Kokkos: C++ library for performance portability**
  Node-level parallelism, parallel pattern and 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 !
- timeframe : ~ 2021 - 2022

Many challenges:

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

Summit

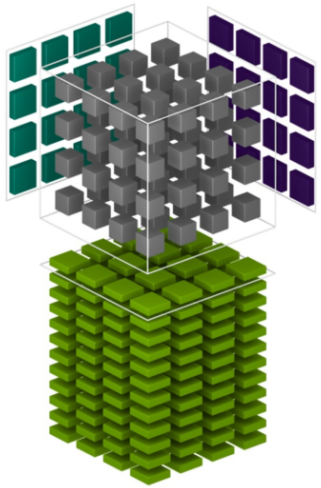
current #1 @top500

Post-K (Japan, 2021 ?)

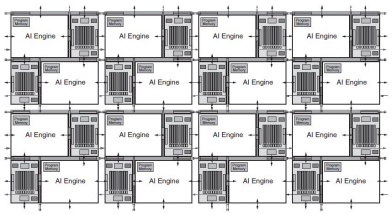
...also about building an economic ecosystem

- 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 (BFLOAT16, for future CooperLake), ...
- **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 - hardware diversity !

- **US**: Summit , Sierra ⇒ mostly OpenPower (IBM P9 + Nvidia V100), GPU-based architecture, #1 and #2 @top500
- **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) ⇒ #3 @top500 , Sunway TaihuLght, 93 PFlops
  - Dhyana, AMD-licenced x86 multicore (300 M$!), identical to AMD EPYC
- **Japan:** **Post K**(Fujitsu, ARM, RIKEN) A64FX ARM (**home grown**, started in 2014, 900 M$), GPU, etc ...
- **Europe ?** lagging behind but new organization EuroHPC (2019), EC H2020 budget (~ 500 M€)
  home grown ARM and/or RISC-V architecture, just starting development

Supercomputers architectures - TOP500

A Supercomputer is designed to be at bleeding edge of current technology.
Leading technology paths (to exascale) using TOP500 ranks (**Nov. 2016**)

- **Multicore:** Maintain complex cores, and replicate (x86, SPARC) (#7, 10)
- **Manycore/Embedded:** Use many simpler, low power cores from embedded (IBM BlueGene) (#4, 9)
- **Manycore/Sunway TaihuLight** (# 1)
- **Manycore/Intel XeonPhi (1st and 2nd gen):** Use many simpler cores with wide SIMD instructions, (# 2, 5, 6)
- **Massively Multithread/ GPU:** (# 3, 8)

Sunway Taihulight : programmed with MPI+OpenACC

Supercomputers architectures - TOP500

A Supercomputer is designed to be at bleeding edge of current technology.
Leading technology paths (to exascale) using TOP500 ranks (**Nov. 2018**)

- **Multicore:** Maintain complex cores, and replicate (x86, SPARC) (#8)
- **Manycore/Embedded:** Use many simpler, low power cores from embedded (IBM BlueGene) (#10)
- **Manycore/Sunway TaihuLight** (# 3)
- **Manycore/Intel XeonPhi (1st and 2nd gen):** Use many simpler cores with wide SIMD instructions, (# 6)
- **Massively Multithread/ GPU:** (# 1, 2, 5, 7, 9)

Sunway Taihulight : programmed with MPI+OpenACC

About DOE Coral next generation computing facility

- As part of **CORAL** (Next gen supercomputers): **Center for Accelerated Application Readiness**
- Provide **programming environments and tools** that enable **portability**

Two Tracks for Future Large Systems

Many Core

- 10's of thousands of nodes with millions of cores
- Homogeneous cores
- Multiple levels of memory – on package, DDR, and non-volatile
- Unlike prior generations, future products are likely to be self hosted

Cori at NERSC

- Self-hosted many-core system
- Intel/Cray
- 9300 single-socket nodes
- Intel® Xeon Phi™ Knights Landing (KNL)
- 16GB HBM, 64-128 GB DDR4
- Cray Aries Interconnect
- 28 PB Lustre file system @ 430 GB/s
- Target delivery date: 2016

Aurora at ALCF

- Self-hosted many-core system
- Intel/Cray
- Intel® Xeon Phi™ Knights Hill (KNH)
- Target delivery date: 2018

Hybrid Multi-Core

- CPU / GPU Hybrid systems
- Likely to have multiple CPUs and GPUs per node
- Small number of very fat nodes
- Expect data movement issues to be much easier than previous systems – coherent shared memory within a node
- Multiple levels of memory – on package, DDR, and non-volatile

Summit at OLCF

- Hybrid CPU/GPU system
- IBM/NVIDIA
- 3400 multi-socket nodes
- POWER9/Volta
- More than 512 GB coherent memory per node
- Mellanox EDR Interconnect
- Target delivery date: 2017

7 IBM Quarterly, LLNL, March 1, 2016

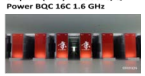
OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

reference a-first-look-at-summit-supercomputer-application-performance (March 2018)

About DOE Coral next generation computing facility

- As part of **CORAL** (Next gen supercomputers): **Center for Accelerated Application Readiness**
- Provide **programming environments and tools** that enable **portability**

Two Tracks for Future Large Systems



Many Core

- 10's of thousands of nodes with millions of cores
- Homogeneous cores
- Multiple levels of memory – on package, DDR, and non-volatile
- Unlike prior generations, future products are likely to be self hosted

Cori at NERSC

- Self-hosted many-core system
- Intel/Cray
- 9300 single-socket nodes
- Intel® Xeon Phi™ Knights Landing (KNL)
- 16GB HBM, 64-128 GB DDR4
- Cray Aries Interconnect
- 28 PB Lustre file system @ 430 GB/s
- Target delivery date: 2016

Aurora at ALCF

- Self-hosted many-core system
- Intel/Cray
- Intel® Xeon Phi™ Knights Hill (KNH)
- Target delivery date: 2018

Retargeted
Exascale (2021)
+ Machine Learning

Hybrid Multi-Core

- CPU / GPU Hybrid systems
- Likely to have multiple CPUs and GPUs per node
- Small number of very fat nodes
- Expect data movement issues to be much easier than previous systems – coherent shared memory within a node
- Multiple levels of memory – on package, DDR, and non-volatile

Summit at OLCF

- Hybrid CPU/GPU system
- IBM/NVIDIA
- 3400 multi-socket nodes
- POWER9/Volta
- More than 512 GB coherent memory per node
- Mellanox EDR Interconnect
- Target delivery date: 2017

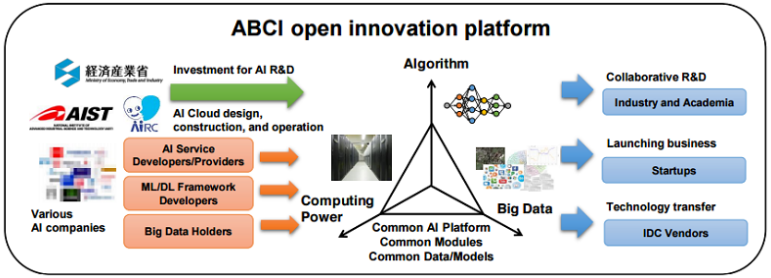
7 IBM Quarterly, LLNL, March 1, 2016

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

reference a-first-look-at-summit-supercomputer-application-performance (March 2018)

HPC architectures - Trends - Who's driving ?

- **Artificial Intelligence** applications : e.g. **Japan** (ABCI: a 130 single precision PetaFlops system in late 2017) for Companies (book time for a fee) **AI Bridging Cloud Infrastructure**: goal is 43 (FP32) GigaFlops/Watt



- **Energy efficiency**, e.g. Nvidia's DGX-1 node server (1 Dual Xeon + 8 GPU P100) aimed at deep learning (~ 18 (FP64) GigaFlops/Watt).

- **Several new hardware solutions** to come next year and after: Intel Knights Mill (XeonPhi, 3rd gen), FPGA (?) for dedicated specific applications, ... ⇒ **a good programing model !**

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

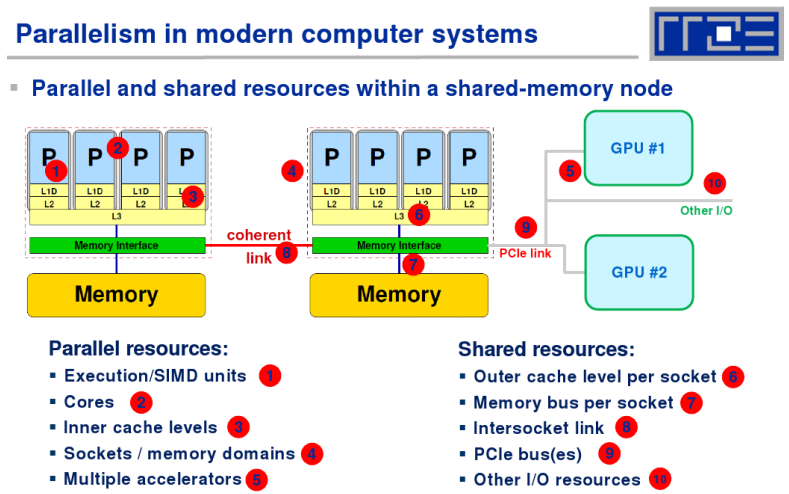


Figure: Multi-core node summary, source: multicore tutorial (SC12) by G. Hager and G. Wellein

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 [1]
- 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

[1] 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 [1]
- 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.

[1] 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**
  ⇒ 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:** OpenCL, CUDA, ...
- **Directive approach (code annotations)** for multicore/GPU, ...:
  - OpenMP 4.5 (Clang, GNU, PGI, ...)
  - OpenACC 2.5 (PGI, 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, Alpaka, Dash-project, agency, ...
  - Cross-platform frameworks
    - Chamm++: message-driven execution, task and data migration, distributed load-balancing, ...
    - hpx (heavy use of new c++ standards (11,14,17): `std::future`, `std::launch::async`, distributed parallelism, ...)
    - SYCL (Khronos Group *standard*), one implementation by CodePlay, by Keryell/Xilinx, ..., parallel STL, wide hardware targets (CPU, GPU, FPGA, ...)
- **Use an embedded Domain Specific Language (DSL)**
  - Halide (for image processing),
  - NABLA (for HPC, developped at CEA, PDE mesh+particules apps)

SC16-compareSYCL-Michael-Wong_Nov16.pdf

Performance portability issue : memory management

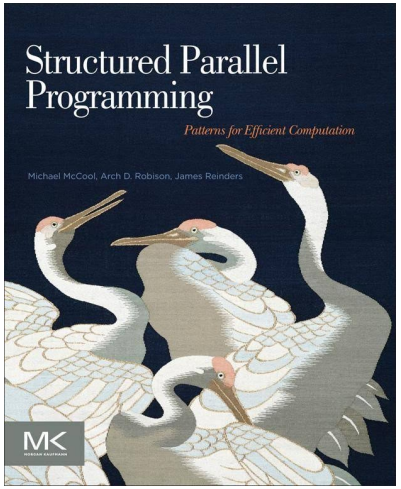
- Right now **directives-based approaches** focus on algorithmic pattern, and less on memory layout (might change in the near future, at least in OpenMP).
- CPU and GPU for example require **different memory layout** for **maximun performance:**
  - vectorization on CPU
  - memory coalescence on GPU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

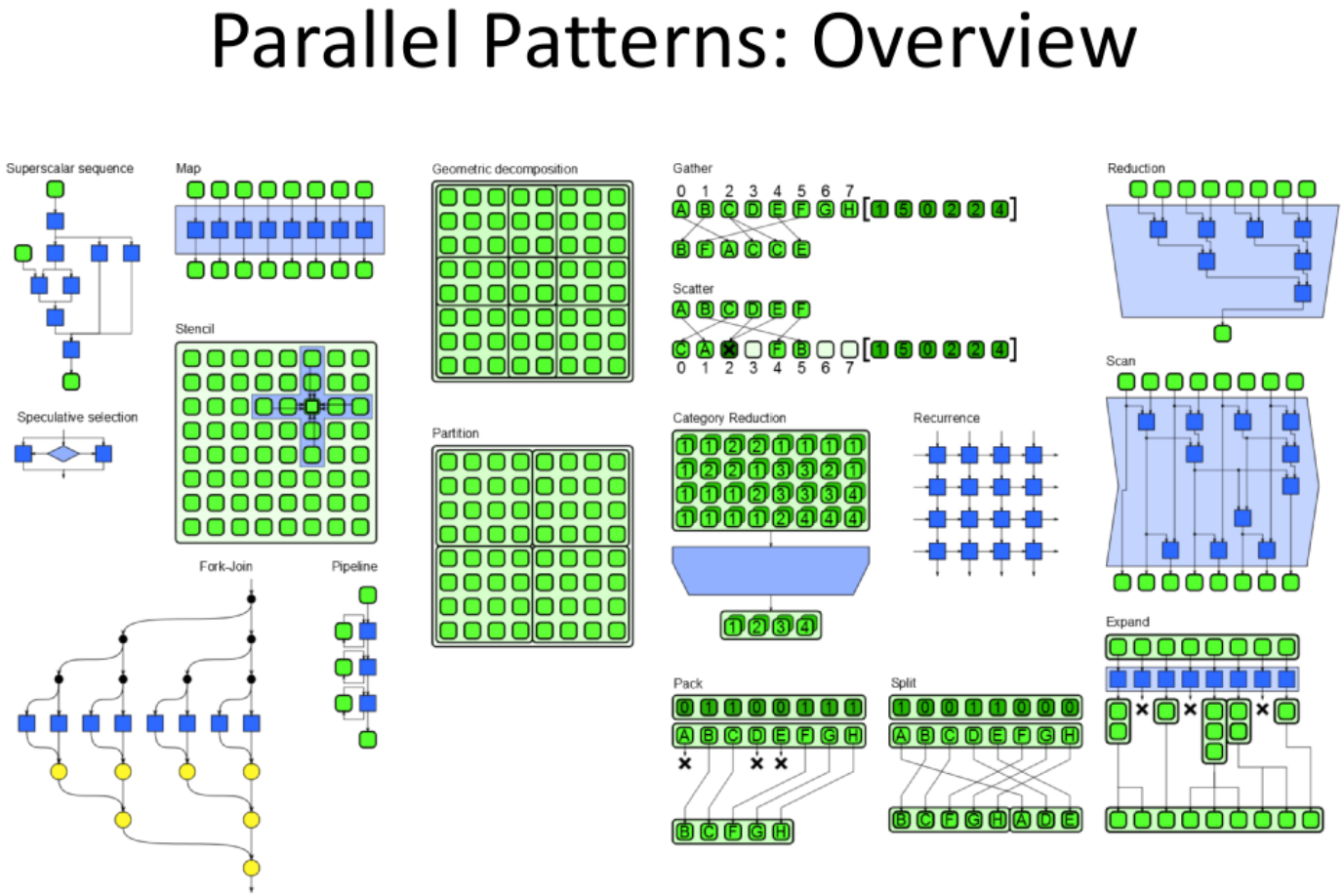
- **pattern** : **a basic structural entity of an algorithm**
- book Structured Parallel Programming: Patterns for Efficient Computation



- 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



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

Future of accelerator programming

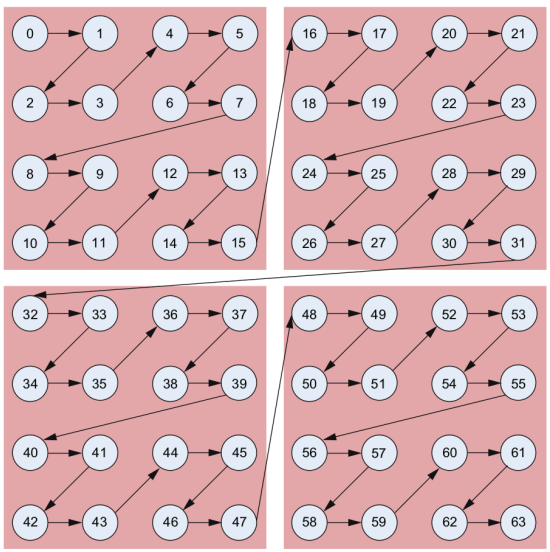
- **passive libraries:** a collection of subroutines
- **active libraires:** take an active role in compilation (specialize algorithms, tune themselves for target architecture).

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	--

HiHAT initiative: Hierarchical Heterogenous Asynchronous Tasking

- for Runtime frameworks developpers + HW vendors
- Tasking frameworks
- Fonctionalities

LANGUAGE OR TASKING FRAMEWORKS

Varying degrees of involvement. Those in bold posted presentation materials.

Some part of each has expressed interest.

- C++ (**CodePlay**, IBM)
- CHARM++ (**UIUC**)
- Darma (**Sandia**)
- Exa-Tensor (**ORNL**)
- Fortran (IBM)
- Gridtools (CSCS, Titech)
- HAGGLE (PNNL/HIVE)
- HPX (CSCS)
- Kokkos, Task-DAG (**SNL**)
- Legion/Realm (**Stanford/NV**)
- OCR (**Intel**, **Rice**, GA Tech)
- PaRSEC (**UTK**)
- Raja (**LLNL**)
- Rambutan, UPC++ (LBL)
- R-Stream (**Reservoir Labs**)
- SyCL (**CodePlay**)
- SWIFT (**Durham**)
- TensorRT (**NVIDIA**)
- VMD (**UIUC**)

Similarly with implementations

- Argobots (**ANL**)
- OpenCL (**CodePlay**)
- Qthreads, NoRMa (**SNL**)
- UCX/UCS (**ARM**)

And a sampling of end users

- ANL, **ANSYS**, Blue Brain (EPFL), CSCS, ECP, GROMACS (KTH), ICIS.PCZ.PL, Lattice Microbes (UIUC),LANL, LBL, **LLNL**, NEMO5 (Purdue), **ORNL**

reference HiHAT_Mini-Summit_17_Overview.pdf

reference `http://slideplayer.com/slide/12990108/`

HiHAT initiative: Hierarchical Heterogenous Asynchronous Tasking

- for Runtime frameworks developpers + HW vendors
- Tasking frameworks
- Fonctionalities

A RETARGETABLE FRAMEWORK

Interfaces are common across multiple targets

Action / service	Description	Example
Computation	Target-specific code isolated in tasks, different implementation for each target, layout	*Invoke task named FOO on target X*
Data layout	Multiple data layouts, with implementations specialized for each	*Data layout Y (vs. Z) is used to select which implemention of FOO to execute*
Data movement	Bring input data for task T to where it executes, send output data to where it's needed	*Fetch FOO's data from wherever it was produced, send its outputs to consumers Optionally re-layout data on the way*
Coordination	Observe and enforce data and control dependences	*FOO doesn't execute until its predecessors complete, the data is sent and formatted*
Scheduling	Select best resources to bind computes and data to and ordering, based on cost models Trade-off across multiple targets, data layouts	*FOO doesn't execute until its predecessors complete, the data is sent and formatted*

reference HiHAT_Mini-Summit_17_Overview.pdf

reference `http://slideplayer.com/slide/12990108/`

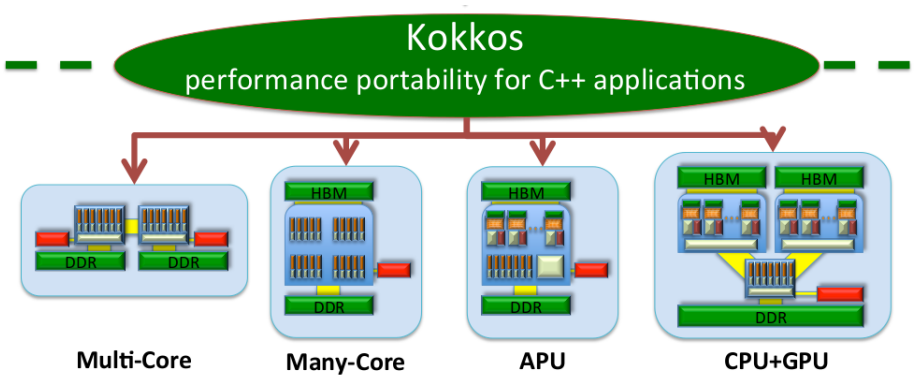
AMT : Asynchonous Many Task frameworks

- Legion (Stanford, C++ Runtime, Regent/language, data partioning, GASNet/PGAs)
- StarPU (Inria Bordeaux, graph of tasks, runtime task scheduling, data dependecies ⇒ task dependencies)
- DARMA (Sandia NL, Async. Many Task programing, abstraction between applications and and low-level runtime scheduling, focus on node-centric scheduling, could be a building block for other)
- Charm++ (Univ. Illinois, Urbana Champain, implements a distributed adaptive runtime system, age > 15 years)
- Uintah, (DAG task graph-based computational framework, PDE solving on structured adaptive grids, scalable IO PIDX, GPU)
- HPX (LSU, general purpose C++ runtime system for parallel and distrib. app.)
- SYCL (cross-platform async task graph, C++/OpenCL, no dist. parallelism yet)
- ParSEC/PLASMA (U. Tennessee, architecture aware scheduling of micro-task on heterogenous hardware, linear algebra applications)
- DASH (distributed data struct. + C++/template PGAs)
- HiCMA (Hierarchical Computations on Manycore Architectures), linear algebra, low-rank approximation

others runtime systems: `https://hihat-wiki.modelado.org/Runtime_Clients`
Comparative analysis of Legion, Charm++, Uintah

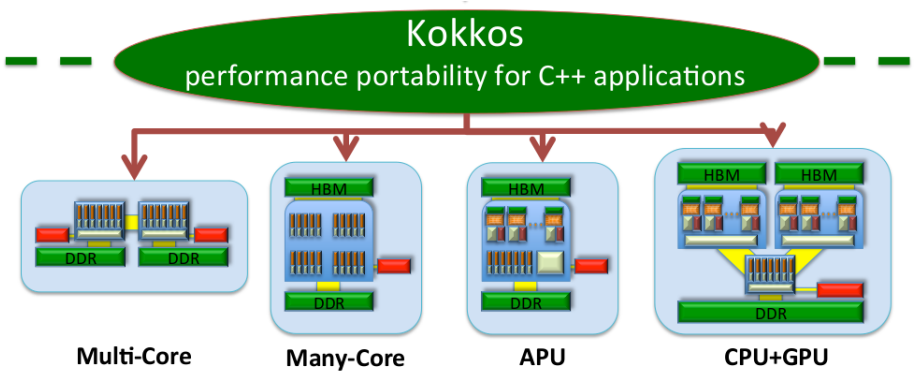
Kokkos: a programming model for performance portability

- **Kokkos** is a **C++ library** with **parallel algorithmic patterns** AND **data containers** for **node-level parallelism**.
- Implementation relies heavily on **meta-programing** to derive native low-level code (OpenMP, Pthreads, CUDA, ...) and adapt data structure memory layout at compile-time
- Core developers at **SANDIA NL** (**H.C. Edwards, C. Trott**)



Kokkos: a programming model for performance portability

- **Open source**, https://github.com/kokkos/kokkos
- Primarily developped as a base building layer for **generic high-performance parallel linear algebra** in Trilinos
- Also used in molecular dynamics code, e.g. LAMMPS
- Goal: **ISO/C++ 2020 Standard** subsumes Kokkos abstractions [2]

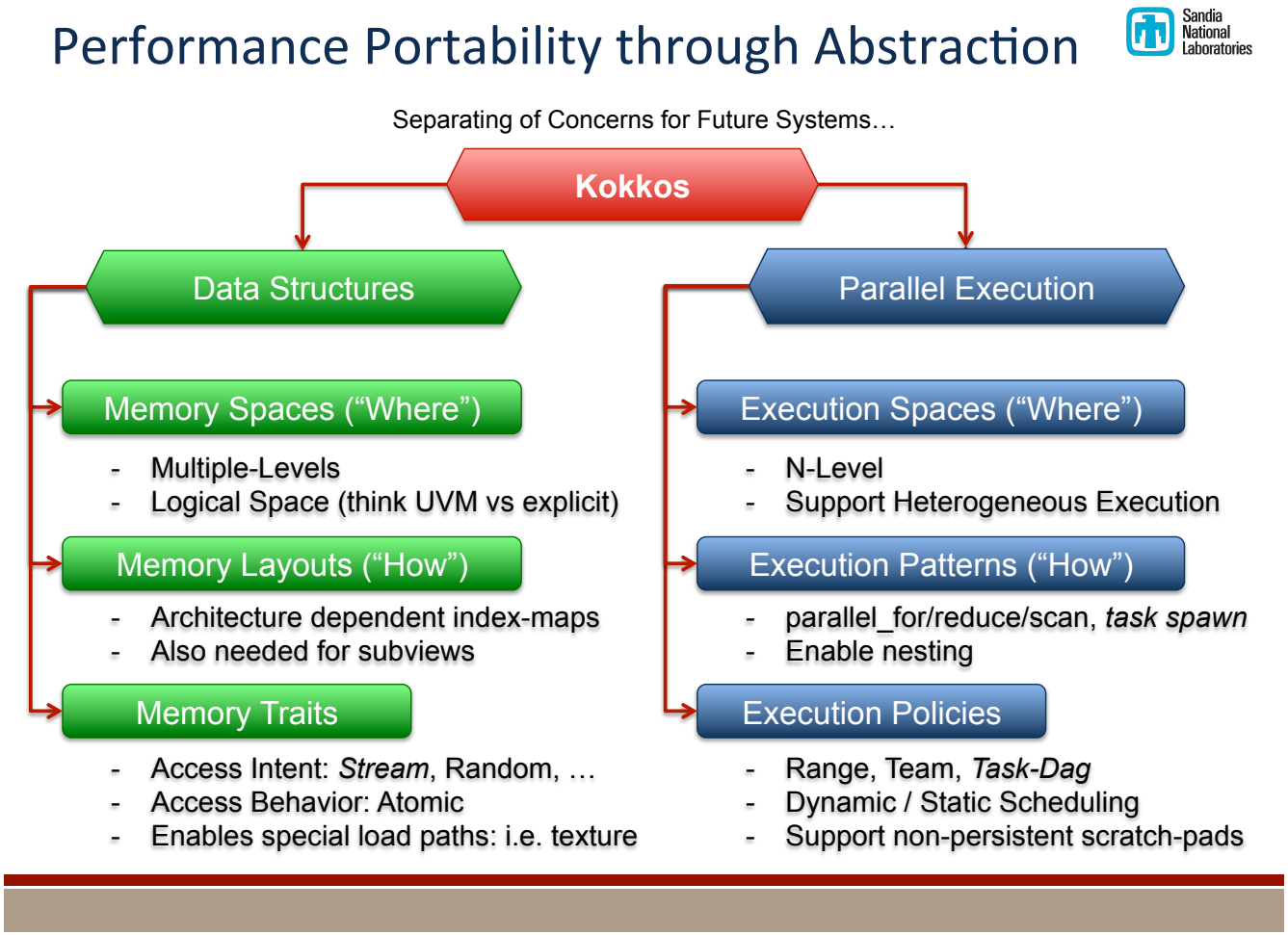


[2] see mdspan proposal https://github.com/kokkos/array_ref

Kokkos abstract concepts

- **Execution patterns (what):**
  parallel_for, parallel_reduce, ...
- **Execution policy (how):**
  range iterations, teams of threads, ...
- **Execution space (where):**
  OpenMP, PThreads, CUDA, numa, ...
- **Memory space: data containers** with architecture adapted memory layout
  `Kokkos::View, Kokkos::DualView, Kokkos::UnorderedMap,...`
- **Memory layout:** (important for vectorization, memory coalescence, ...)
  row-major, column-major, AoS, SoA, ...
  $data(i,j,k)$ architecture aware.

reference: Kokkos: Manycore programmability and performance portability, SIAM conference, Paris, 2016



reference:

https://cfwebprod.sandia.gov/cfdocs/CompResearch/docs/Kokkos-Multi-CoE.pdf

Kokkos: Sparse Matrix-Vector Multiply

- **Baseline serial version**

```
for ( int i = 0 ; i < nrow ; ++i ) {
  for ( int j = irow[i] ; j < irow[i+1] ; ++j )
    y[i] += A[j] * x[ jcol[j] ];
}
```

- **Simple Kokkos parallel version**

```
parallel_for( nrow , KOKKOS_LAMBDA( int i ) {
  for ( int j = irow[i] ; j < irow[i+1] ; ++j )
    y[i] += A[j] * x[ jcol[j] ];
});
```

- Execution pattern: **parallel for**
- Execution policy: **range iteration**
- Execution space: default (defined at compiled time)
- **Work to do can be**
  - A **Lambda *anonymous* function**, convenient for short loop bodies
  - A **C++ class functor**, maximun flexibility

MiniMD used to bench thread-scalable algorithm before integrating them in LAMMPS (2014)

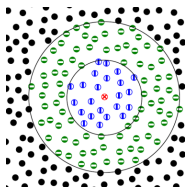
MiniMD Performance

Lennard Jones force model using atom neighbor list

Sandia National Laboratories

- **Solve Newton's equations for *N* particles**
- **Simple Lennard Jones force model:** $F_i = \sum_{j, r_{ij} < r_{cut}} 6\varepsilon \left[\left(\frac{\varsigma}{r_{ij}} \right)^7 - 2 \left(\frac{\varsigma}{r_{ij}} \right)^{13} \right]$
- **Use atom neighbor list to avoid N^2 computations**

```
pos_i = pos(i);
for( jj = 0; jj < num_neighbors(i); jj++) {
  j = neighbors(i,jj);
  r_ij = pos_i - pos(j); //random read 3 floats
  if ( |r_ij| < r_cut )
    f_i += 6*e*( (s/r_ij)^7 - 2*(s/r_ij)^13 )
}
f(i) = f_i;
```



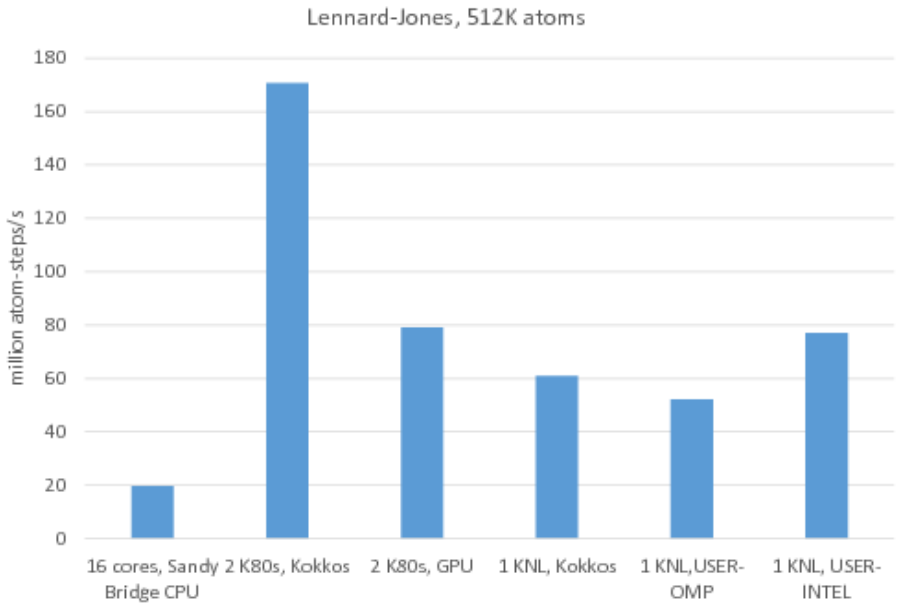
- **Moderately compute bound computational kernel**
- **On average 77 neighbors with 55 inside of the cutoff radius**

17

source: http://lammps.sandia.gov/bench.html
LAMMPS Accelerator benchmarks for CPU, GPU, KNL Oct 2016

Future of accelerator programming: Kokkos among other

MiniMD used to bench thread-scalable algorithm before integrating them in LAMMPS (2014)

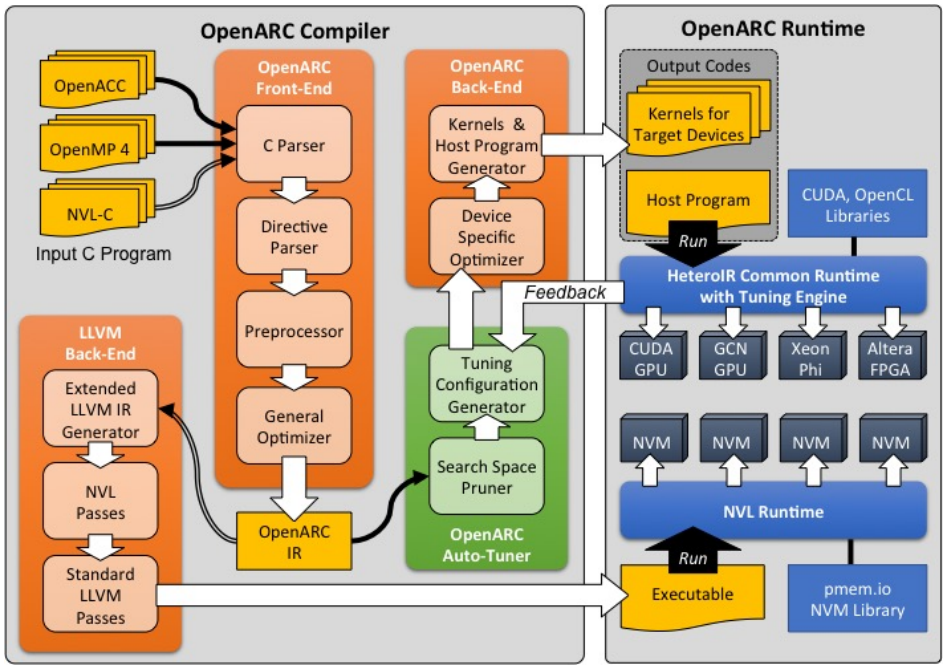


source: `http://lammps.sandia.gov/bench.html`
LAMMPS Accelerator benchmarks for CPU, GPU, KNL Oct 2016

Additionnal links

- CORAL-benchmarks : CORAL Benchmark codes
- DOE-COE-Mtg-2016 / DOE-COE-Mtg-2017 : DOE meeting on performance portability
- https://www.hpcwire.com/2016/04/19/compilers-makes-performance-portable/ : Compilers and More: What makes performance portable, Michael Wolfe (HPCWire article).
- https://github.com/brycelelbach/2016_berkeley_cpp_summit_presentations
- DawnCC (http://cuda.dcc.ufmg.br/dawn/) : automatic parallelization of Code (C/C++), automatic insertions of OpenMP/OpenACC annotations, based on LLVM framework for IR analysis

An interesting research compiler multi-platform



source: http://ft.ornl.gov/research/openarc