Using high-level programming tools aiming at performance portability

A short overview of some C++-based programing model(s) for performance portability

Pierre Kestener[1]

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

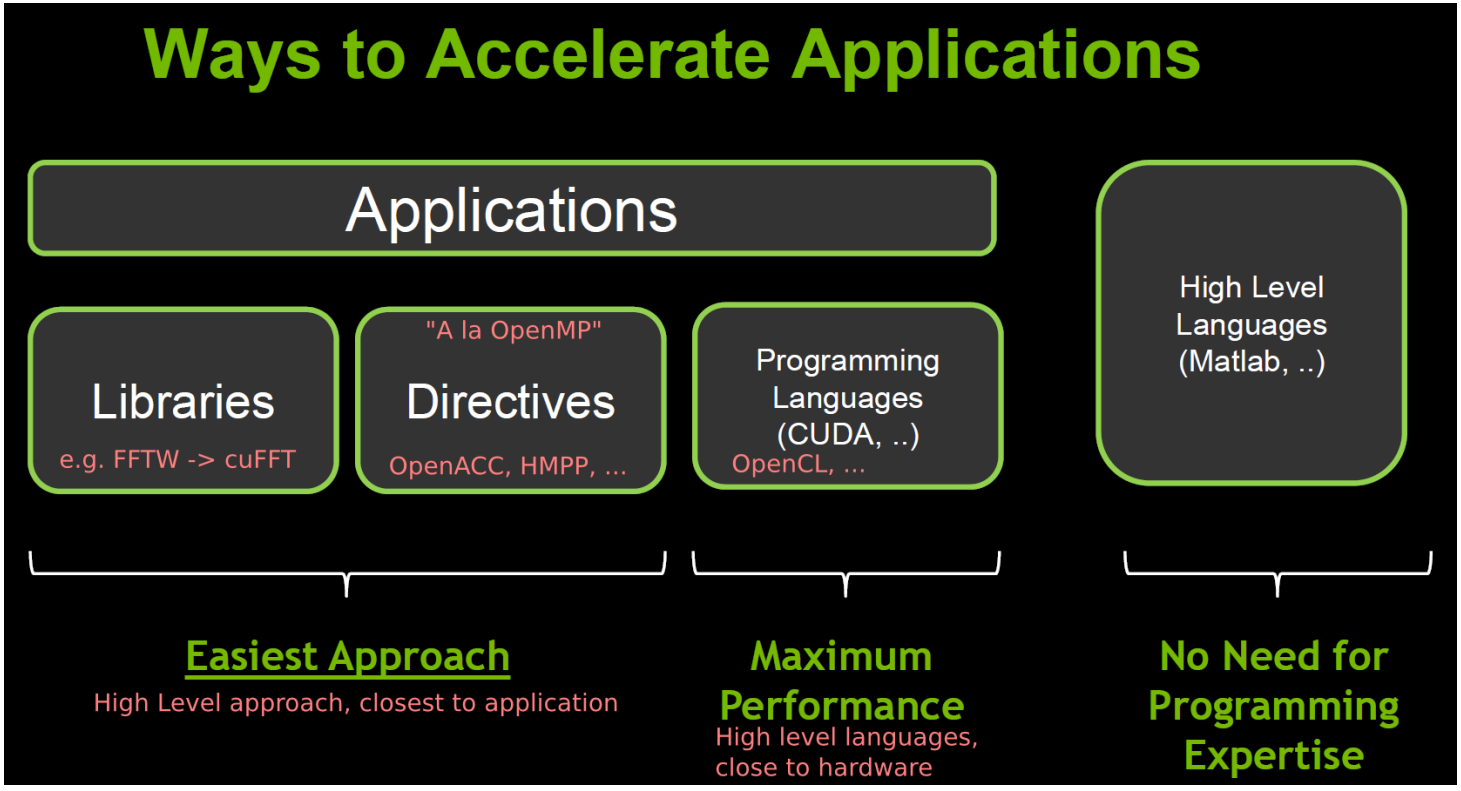
December 19, 2016

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.
- A real life example: **code RamsesGPU** **(high-Mach number turbulent MHD)** (partially) rewritten with Kokkos.

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**.

Summary

1. Introduction
2. Performance Portability
   - Directives: OpenACC / OpenMP
   - (Active) libraries
3. Kokkos introduction
   - Kokkos basics
   - Case study: RamsesGPU on Pascal P100
   - Additionnal slides

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** (# 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
Next year, we might have supercomputers build with **ARMv8** CPU (From China, Japan, US,...), **DOE Coral machines** (**NVidia GPU+IBM Power9, Intel KNL**), ...

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


Tianhe-2 (NUDT): TH-IVB-FEP
Intel Xeon E5-2692 12 C 2.2 GHz
TH Express-2
Intel Xeon Phi 31S1P


Titan (Cray): Cray XK7
AMD Opteron 6274 16C 2.2 GHz
Cray Gemini
NVIDIA K20x


Sequoia (IBM): BlueGene/Q
Power BQC 16C 1.6 GHz


K computer (Fujitsu)
SPARC64 VIIIfx 2.0 GHz
Tofu


Mira (IBM): BlueGene/Q
PowerPC A2 16C 1.6 GHz


Piz Daint (Cray): Cray XC30
Intel Xeon E5-2670 8C 2.6 GHz
Cray Aries
NVIDIA K20x

Edison (Cray): Cray XC30
Intel Xeon E%-2695v2 12C 2.4 GHz
Aries

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

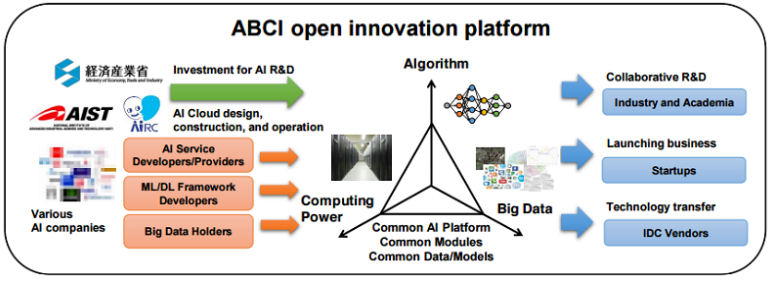
- Hybrid CPU/GPU system
- IBM/NVIDIA
- 3400 multi-socket nodes
- POWER9/Volta
- More than 512 GB coherent memory per node
- Mellanox EDR Interconnect
- Target delivery date: 2017

7 IBM Quarterly, LLNL, March 1, 2016

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

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 !**

Supercomputer node architecture

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

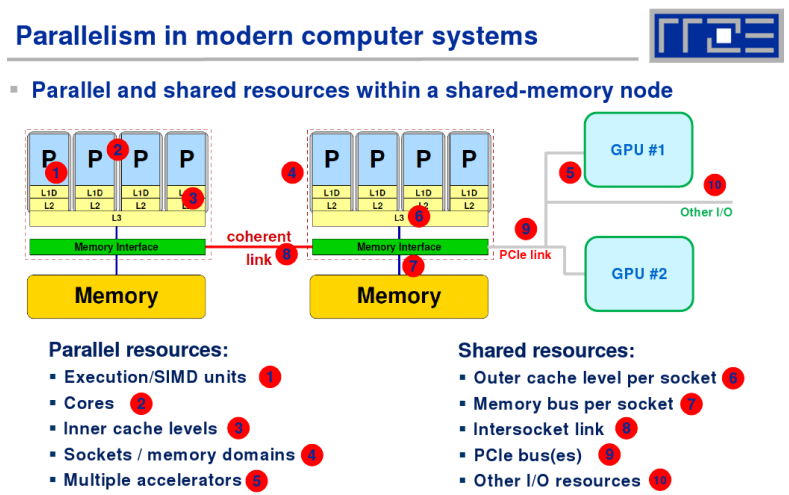


Figure: Multi-core node summary, source: multicore tutorial (SC12) by G. Hager and G. Wellein

Summary

1 Introduction

2 Performance Portability
- Directives: OpenACC / OpenMP
- (Active) libraries

3 Kokkos introduction
- Kokkos basics
- Case study: RamsesGPU on Pascal P100
- Additionnal slides

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
- **High Developper / programmer productivity**

[1] source: Matt Norman, WACCPD 2016

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, ...
  - 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, ...)
- **Use an embedded Domain Specific Language (DSL)**
  - Halide (for image processing),
  - NABLA (for HPC, developped at CEA, PDE mesh+particules apps)

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**

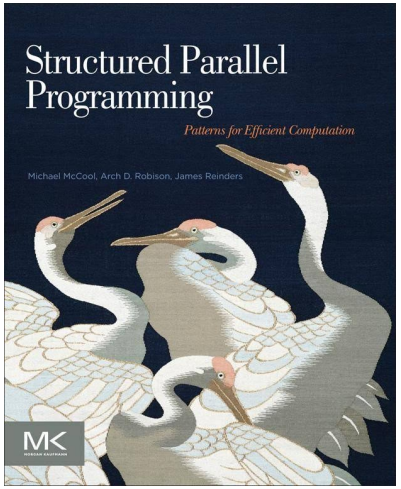
Motivation: Variety in Memory Hierarchies

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	✓	✓	✓	-	-	✓	✓	-

*Other names and brands may be claimed as the property of others.
© 2016 Intel Corporation

Programming with structured parallel patterns

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



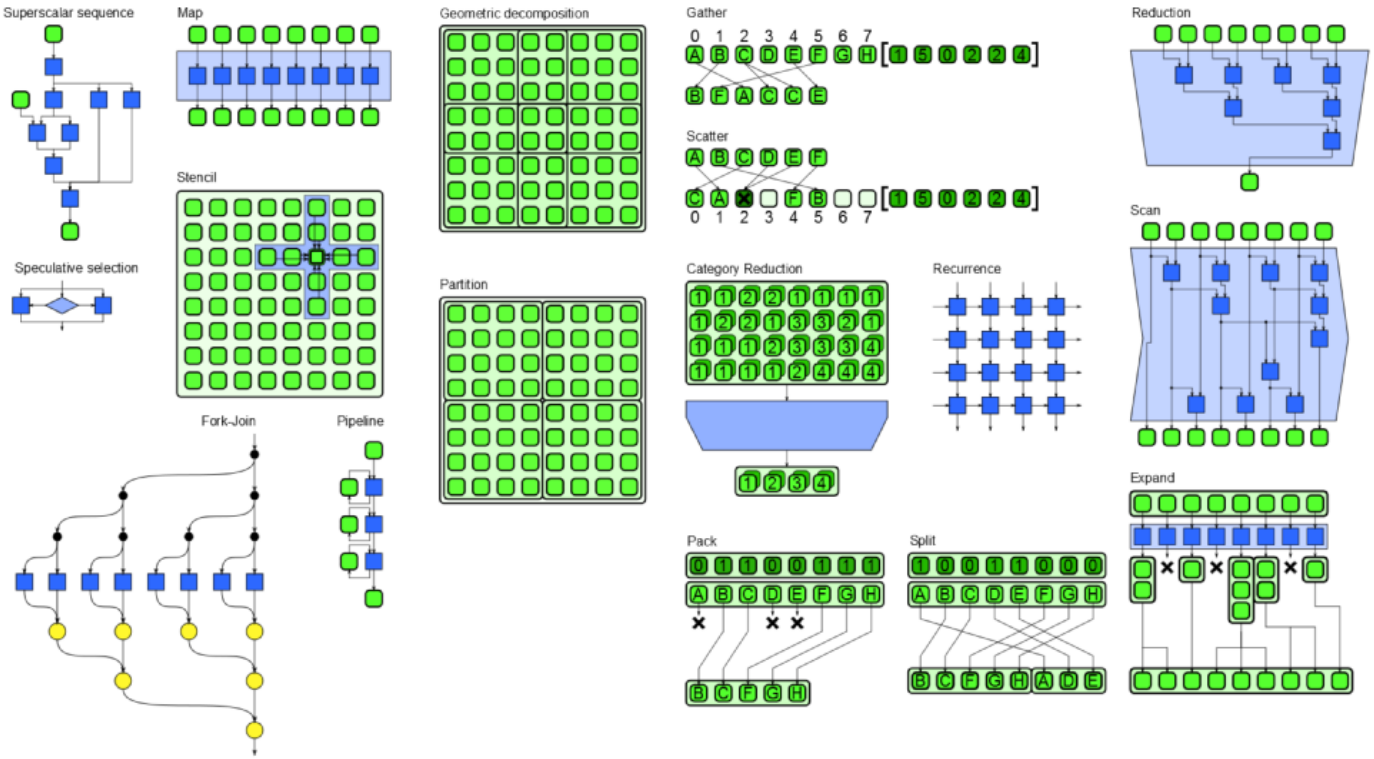
- implementation: Intel TBB, OpenMP, OpenACC and many others
- OpenMP/OpenAcc for GPU/XeonPhi: pattern-based comparison: map, stencil, reduce, scan, fork-join, superscalar sequence, parallel update

reference:

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

Programming with structured parallel patterns

Parallel Patterns: Overview



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

Future of accelerator programming

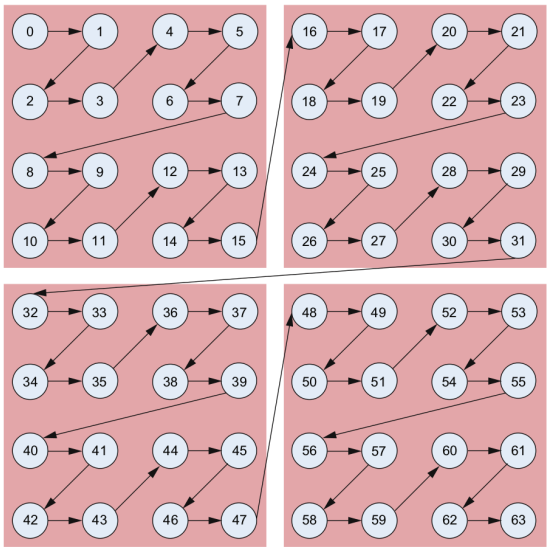
- **passive libraries:** a collection of subroutines
- **active libraires:** take an active role in compilation (specialize algorithms, tune themselves for target architecture).

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	--

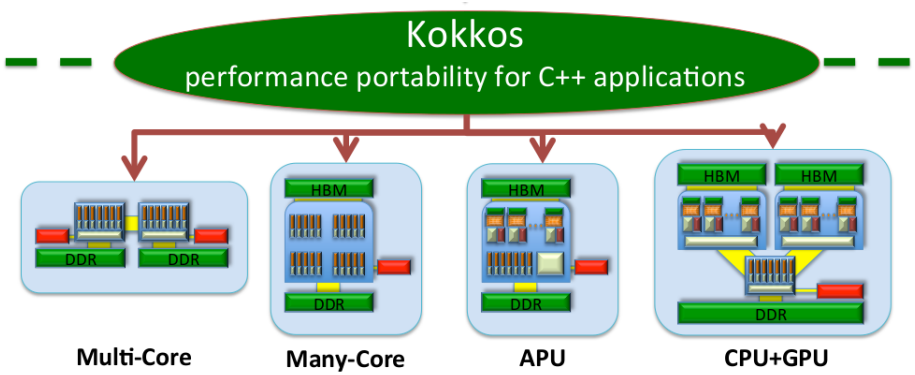
dim	naïve 244 thr	Morton 244 thr	speedup
256	0.02	0.003	6.67
512	0.26	0.008	32.5
1024	1.78	0.046	38.7
2048	12.87	0.4	32.18
4096	105.5	2.9	36.38
8192	105.5	23	36.74
16384	6597	181	36.4
32768	Too long	1468	--

Summary

1. Introduction
2. Performance Portability
   - Directives: OpenACC / OpenMP
   - (Active) libraries
3. Kokkos introduction
   - Kokkos basics
   - Case study: RamsesGPU on Pascal P100
   - Additionnal slides

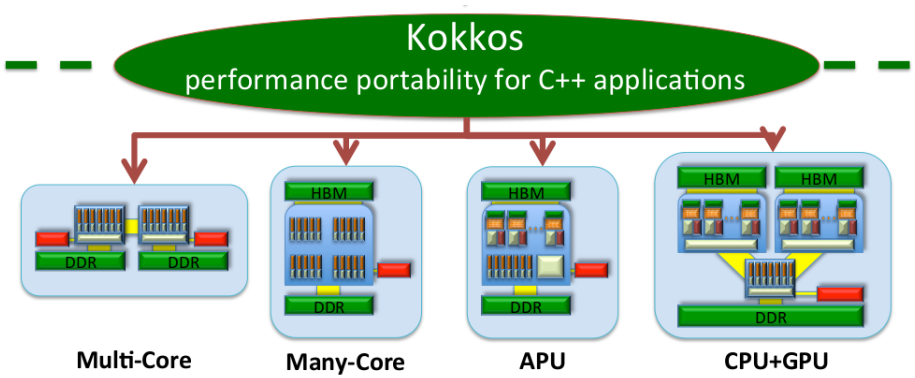
Kokkos: a programming model for performance portability

- **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



Kokkos: a programming model for performance portability

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

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

Future of accelerator programming: Kokkos among other

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

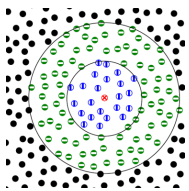
MiniMD Performance

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;
```



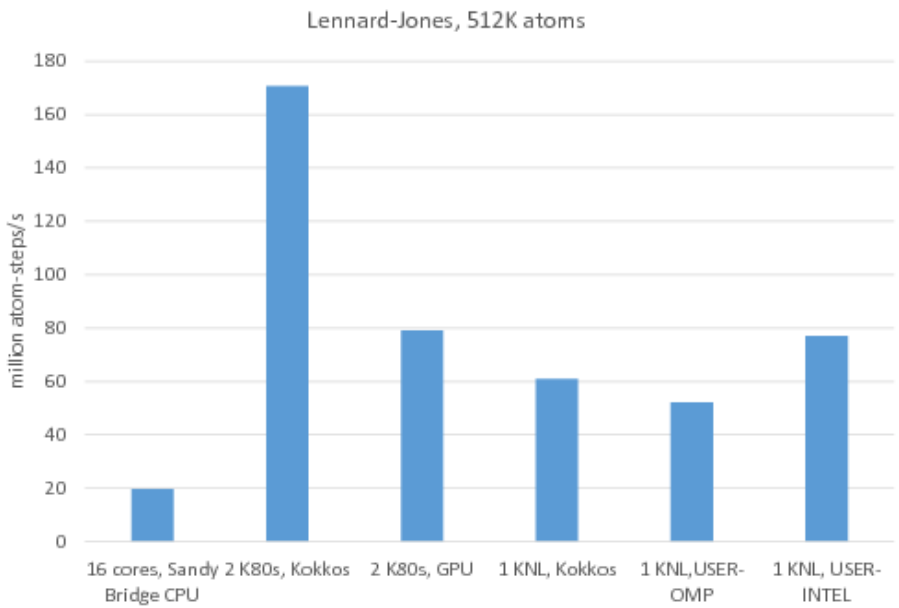
- **Moderately compute bound computational kernel**
- **On average 77 neighbors with 55 inside of the cutoff radius**

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)

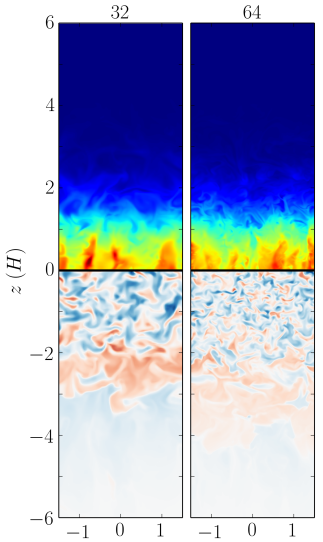


source: `http://lammps.sandia.gov/bench.html`
LAMMPS Accelerator benchmarks for CPU, GPU, KNL Oct 2016

RamsesGpu with Kokkos

- RamsesGPU is a C++/Cuda code for Compressible MHD application, developped since 2009-2010 (ran on Titane Tesla S1070, sm_13).
- Several numerical schemes variants, ~ 70 CUDA kernels.
- **MagnetoRotational Turbulence in accretion disks**; 576-nodes jobs (K20) on NCSA BlueWaters, collegues from CEA and Univ. Illinois: Ryan, Gammie, Fromang, P.K.
- Recently, rewrite core kernel application using **Kokkos**.

$$
\begin{aligned}
\frac{\partial \rho}{\partial t} &= -\nabla \cdot (\rho \boldsymbol{v}) \,, \\
\frac{\partial \rho \boldsymbol{v}}{\partial t} &= -\nabla \cdot (\rho \boldsymbol{v}\boldsymbol{v}) + (\boldsymbol{B} \cdot \nabla)\, \boldsymbol{B} - \nabla \left(\frac{\boldsymbol{B} \cdot \boldsymbol{B}}{2} + P \right) \\
&\qquad - 2\,\rho\,\Omega_0 \hat{\boldsymbol{e}}_z \times \boldsymbol{v} + \rho \nabla \left(-\frac{3}{2}\Omega_0^2 x^2 + \frac{1}{2}\Omega_0^2 z^2 \right) \\
\frac{\partial \boldsymbol{B}}{\partial t} &= \nabla \times (\boldsymbol{v} \times \boldsymbol{B}) \,,
\end{aligned}
$$



RamsesGpu with Kokkos

Directionally splitted Hydro - 3D performances

Number of 10^6 cell updates / second versus domain size

taille	M2090 SP/fast	M2090 SP	M2090 DP	K20 SP/fast	K20 SP	K20 DP
32x32x32	18.8			26.6 (+41%)		
64x64x64	83.4			95.2 (+14%)		
96x96x96	100.7			175.1 (+73%)		
128x128x128	114.7	32.1	9.2	**178.7** (+55%)	**72.4**	**34.9**
192x192x192	133.0			226.7 (+70%)		
225x225x225	137.2	39.5	11.1	210.5 (+53%)	97.4	40.6

- Architecture **Kepler K20** *versus* **Fermi M2090**
- Rebuild application with CUDA 5.5 toolchain for architecture 3.5 and tune flags
- Tune max register count for Kepler, and care about *read-only data cache* ==> **No more register spilling, DP perf is optimal !**
- in DP : Kepler is ~ 3.5× **faster** than than M2090

20 / 44

- With Kokkos, from Kepler K80 ⇒ Pascal P100, **performance scaling almost perfect** (~ ×3.0); no tuning required ; 360 Mcell-update/s
- With hand-written CUDA, tuning is required to recover this perf scaling
- Number of lines of codes divided by 2-3
- Get for free an efficient OpenMP implementation

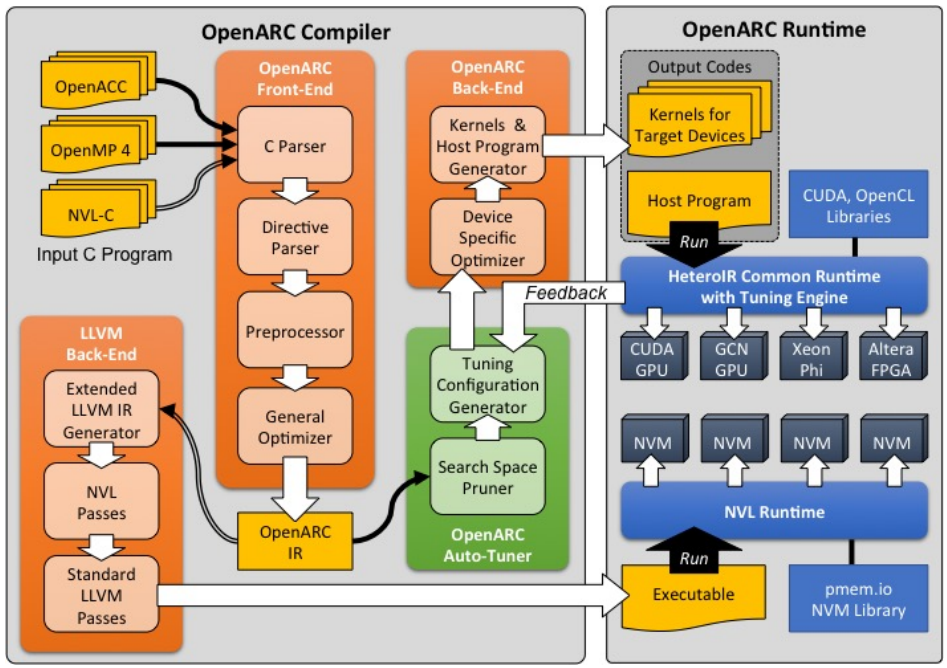
Want to know more ?

- A free 3-days training on **performance portability**:
  Performance portability for GPU applications using high-level programming approaches
  `https://events.prace-ri.eu/event/568/`
- Themes: OpenACC / Kokkos
- Dates: 16-18 January 2017
- Location: IDRIS Computing center, Orsay
- Hardware platform: Ouessant (IBM Power8 + Nvidia P100)

Additionnal links

- https://asc.llnl.gov/CORAL-benchmarks/ : CORAL Benchmark codes
- https://asc.llnl.gov/DOE-COE-Mtg-2016/ : DOE meeting on performance portability
- https://www.hpcwire.com/2016/04/19/compilers-makes-performan : Compilers and More: What makes performance portable, Michael Wolfe (HPCWire article).

An interesting research compiler multi-platform



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