Modèles de programmation associés à la portabilité de performance
Evaluation de la bibliothèque C++/Kokkos

Pierre Kestener

CEA Saclay, DRF, Maison de la Simulation, FRANCE

23 mai 2019

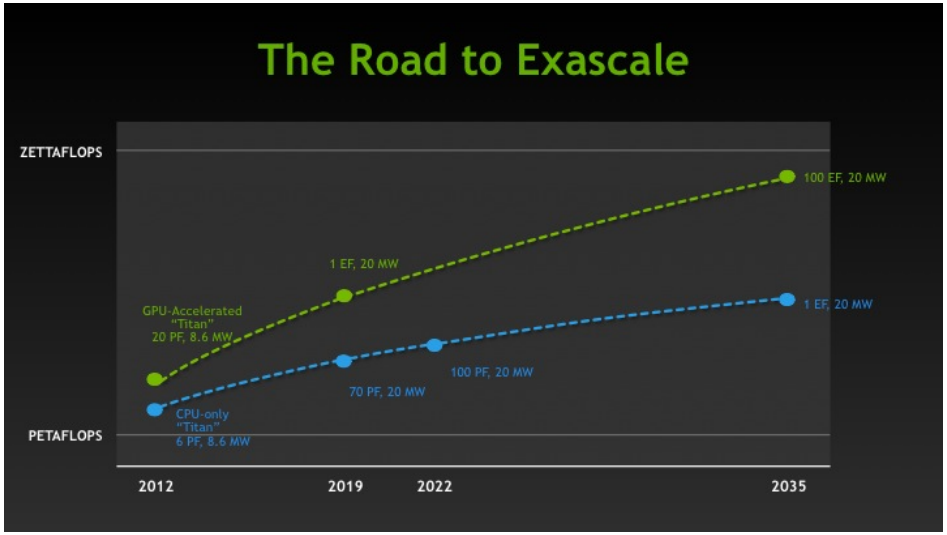
Séminaire Aristore, En route vers l'exascale

Content

- **(Pre-)Exascale trends**
  - **hardware architecture diversity**,
  - need for software programming solutions: **performance portability**
- **Performance portables solutions (for hydrodynamics apps.)**
  - Existing performance portable solutions
  - Short overview of **C++ Kokkos library**: **designing portable parallel codes**
  - Designing Hydrodynamics applications with **Kokkos** (SDM), **perf. measurements (Intel Skylake, Intel KNL, Nvidia GPU, ARM ThunderX2)**

Exascale : days of future past...

- At SC2011, Nvidia CEO talk:



- Document The International Exascale Software Project Roadmap by Dongarra et al. (DOI:10.1177/1094342010391989)
  1. Make a thorough assessment of needs, issues and strategies,
  2. Develop a coordinated software roadmap,
  3. Encourage and facilitate collaboration in education and training.
- **Artificial Intelligence (AI) was not even in the scope...**

Exascale race: technological challenge(s)

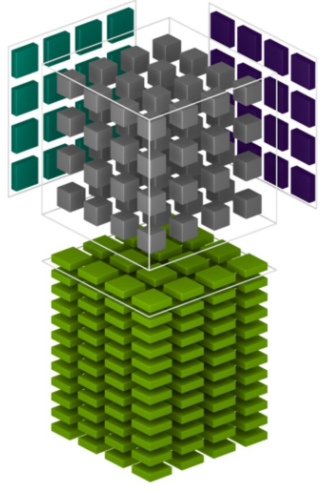
Many technical challenges...

- **Most challenging constraint:** fitting the **electrical power envelop** ($P \in [20-40]$ MW)
- hardware technologies (**interconnect, storage, processor architectures,...**)

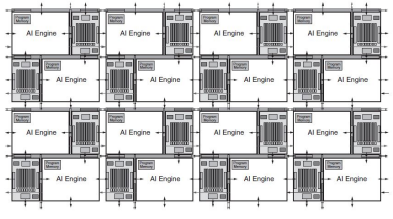
...also about building an economic ecosystem

- **New architectures** for **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 - architecture diversity !

- **US**: Summit , Sierra ⇒ mostly OpenPower (IBM P9 + Nvidia V100), GPU-based architecture, #1 and #2 @top500; exascale machines announced
  - Aurora (Argonne NL): Intel Xe GPU
  - Frontier (Oak Ridge NL): AMD EPYC + Radeon 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) ⇒ #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, early stage

Performance portability and software productivity

- **Developing / maintaining** a **separate implementation** of an application for each **new hardware platform** (Intel KNL, Nvidia GPU, ARMv8, ...) **requires lots of efforts**
- **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 ?

  – high level of abstraction for the end user/developper
  – low-level for hardware optimization

- **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 https://performanceportability.org/
- **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 and software productivity

ARCHITECTURAL DETAILS

How much architectural detail must be revealed to programmers?

Conceal – Hide the feature

instruction-level parallelism

Virtualize – Reveal existence, hide details

vectorized loops

Reveal – All details revealed to programmer

SIMD intrinsics

PGI

6 NVIDIA

source : M. Wolfe, PGI/Nvidia, DOE Perf. Port. Meeting, April 2019

Performance portability and software productivity

THREE EX'S OF PERFORMANCE PORTABILITY

How much detail must the programmer know in order to …

Expose – in the choice of algorithm and data structure

Express – in the choice of program constructs

Exploit – in the choice of how to schedule

PGI

7 NVIDIA

source : M. Wolfe, PGI/Nvidia, DOE Perf. Port. Meeting, April 2019

Performance portability issue

Developper productivity versus optimization

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

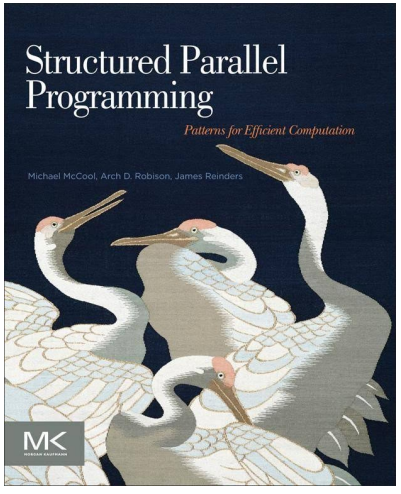
Performance portability issue : algorithmic patterns

- **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++, 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, ...), Intel OneAPI (Q4 2019)
- **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

Programming with structured parallel patterns

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

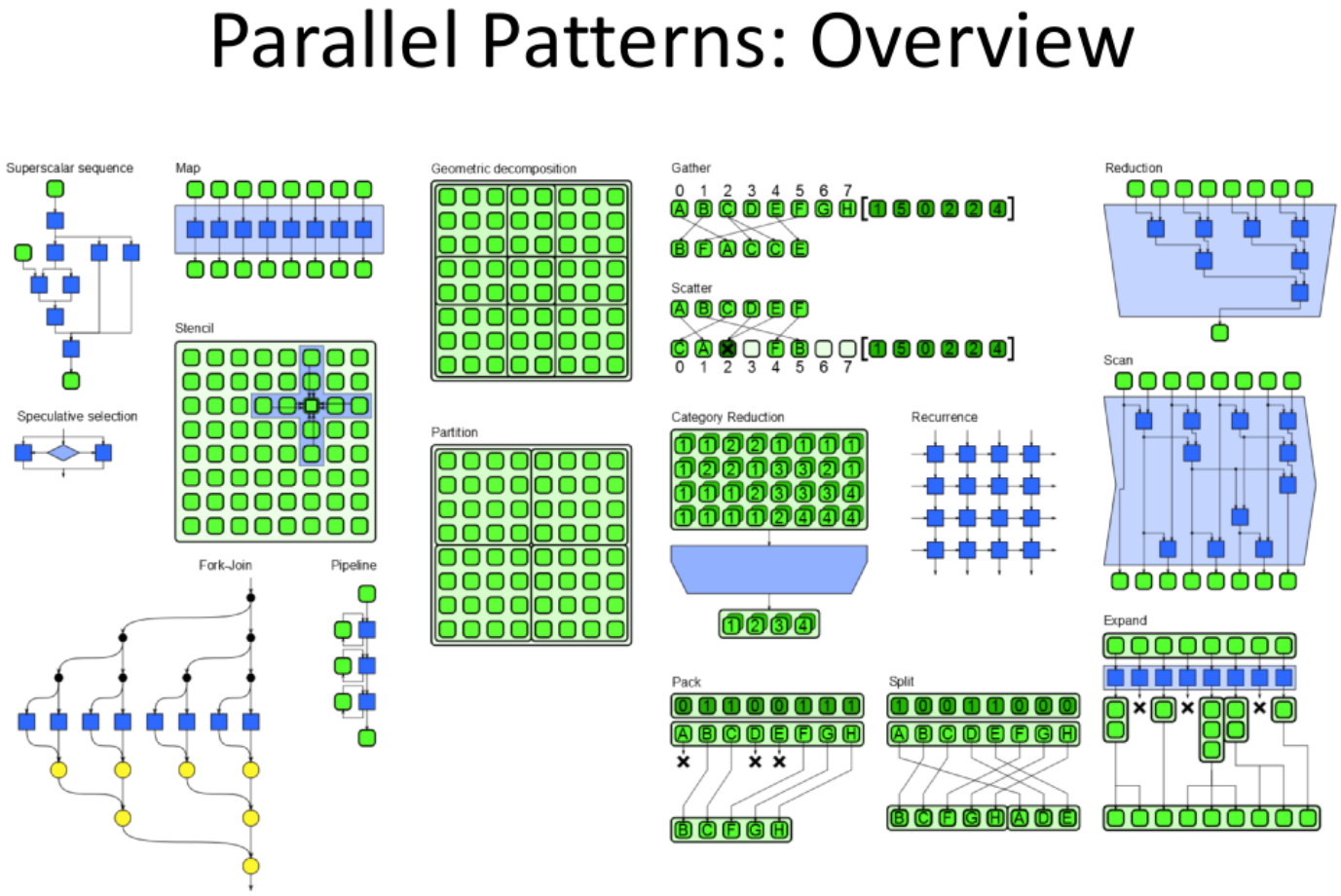


- implementation: Intel TBB, OpenMP, OpenACC and many others
- OpenMP/OpenAcc for GPU/XeonPhi: pattern-based comparison: map, stencil, reduce, scan, fork-join, superscalar sequence, parallel update

reference:

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

Programming with structured parallel patterns

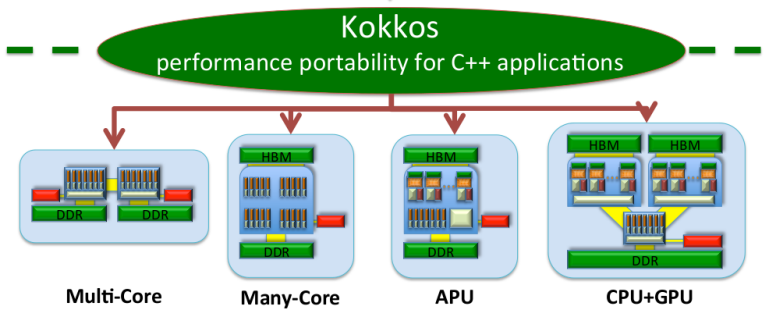


reference: Structured Parallel Programming with Patterns, SC13 tutorial, by M. Hebenstreilt, J. Reinders, A. Robison, M. McCool

Kokkos: a programming model for perf. portability

- **Kokkos** is a **C++ library** for **node-level parallelism** (i.e. **shared memory**) providing:
  - **parallel algorithmic patterns**
  - **data containers**
- 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** [2], **C. Trott**)

Goal: **ISO/C++ 2020 Standard** subsumes Kokkos abstractions
Make Kokkos a sliding window of future c++ features



see mdspan proposal https://github.com/kokkos/array_ref

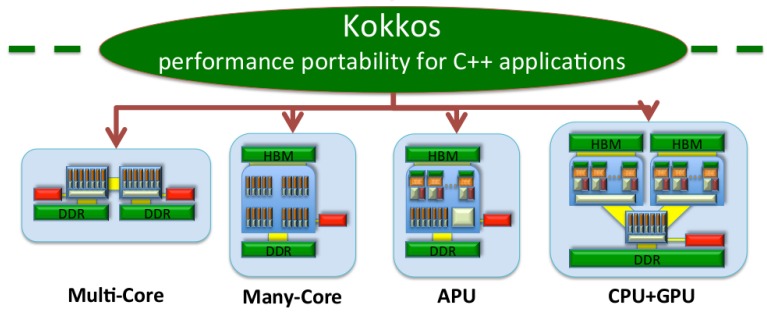
[2] now working @Nvidia

Kokkos: a programming model for perf. portability

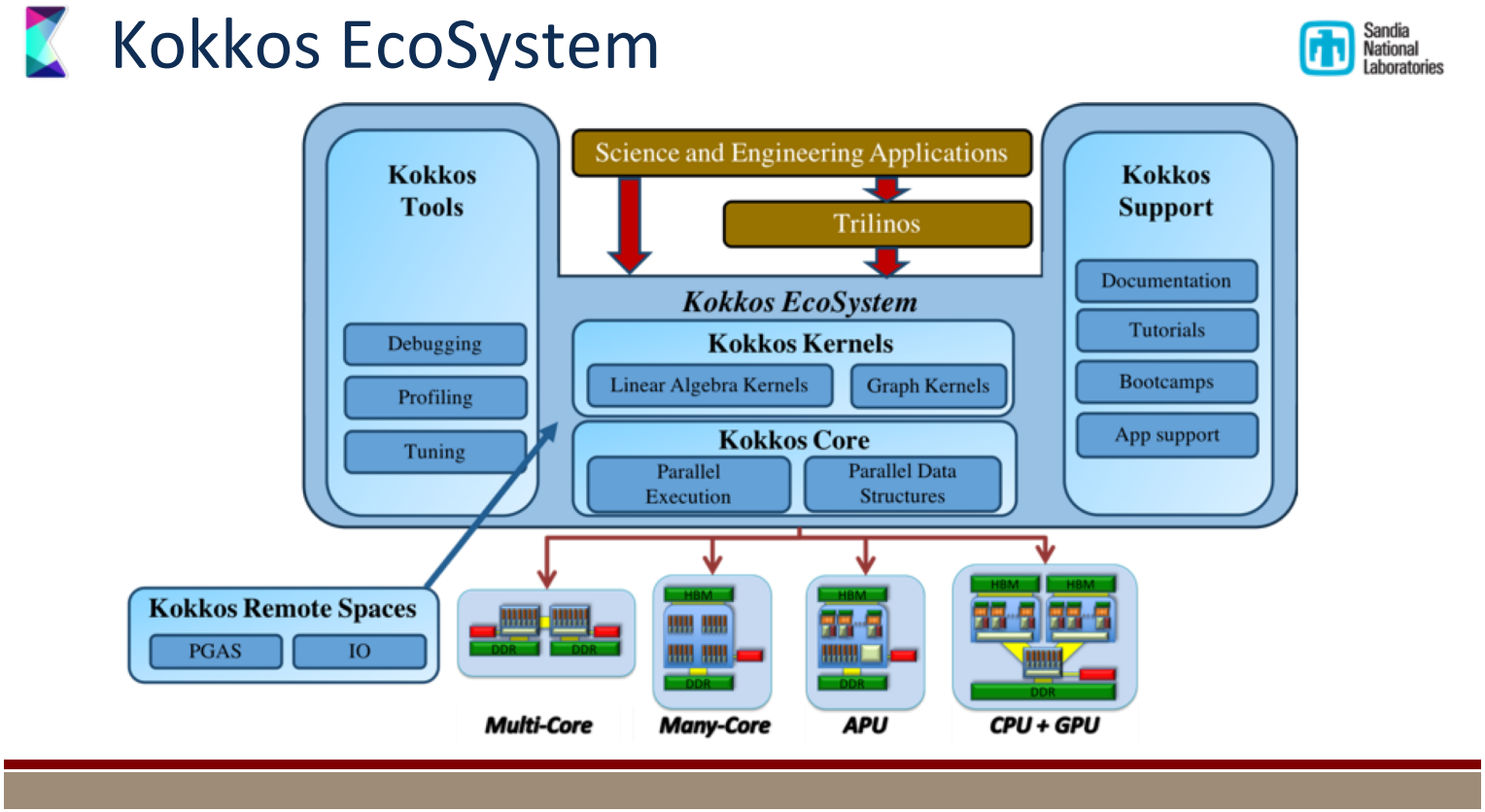
- **Open source**, https://github.com/kokkos/kokkos
- Primarily developped as a base building layer for **generic high-performance parallel linear algebra** in Trilinos
- Also used in
  - LAMMPS (molecular dynamics code),
  - NALU CFD (low-Mach wind),
  - SPARTA/DSMC (rarefied gas flow), SPARC (CFD, RANS, LES, hypersonic flow)
  - Albany (fluid/solid,...)
  - Uintah (structured AMR, combustion, radiation)

Goal: **ISO/C++ 2020 Standard** subsumes Kokkos abstractions
Make Kokkos a sliding window of future c++ features

see mdspan proposal https://github.com/kokkos/array_ref



Kokkos: a programming model for perf. portability



source: C. Trott, DOE Perf. Port. Meeting, April 2019

C++ Kokkos library summary

- Framework for efficient **node-level parallelism (CPU, GPU, ...)**
- Provides
  - **Computationnal parallel patterns** (for, reduce, scan, ...)
  - **Hardware aware memory containers**: e.g. **A multi-dimensionnal data container with hardware adapted memory layout**
- Mostly a header library (C++ metaprograming)

C++ Kokkos library summary

- **What does it mean hardware aware memory containers ?**
- Most commonly in a C/C++, **multi-dimensionnal array access** is done through **index linearization** (row or column-major in 2D):

$$\mathbf{index} = \mathbf{i} + nx * \mathbf{j} + nx * ny * \mathbf{k}$$

- **Fortran** (column-major format) vs **C/C++** (row-major format)
- There is no reason to favour one layout versus the other
  - column-major is better for vectorization on CPU architecture
  - row-major is better for high througput architecture e.g. GPU (memory coalescence)
  - $\Rightarrow$ **Kokkos allows to chose memory layout at compile time**
- In Kokkos, one should/must avoid this index linearization at the user level, let `Kokkos::View` do this job (**decided at compile-time, hardware adapted**)

$$\mathrm{data}(i, j, k)$$

Multidimensional array and data parallelism

Memory layouts / index linearization: e.g. 2D

row-major

$n_x(n_y-1)$	$n_x(n_y-1)+1$	$\cdots$	$n_x n_y - 1$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$2n_x$	$2n_x+1$	$\cdots$	$3n_x-1$
n_x	n_x+1	$\cdots$	$2n_x-1$
0	1	$\cdots$	n_x-1

$$\text{index} = i + n_x j,$$

left layout

fast index on the left

column-major

n_y-1	$2n_y-1$	$\cdots$	$n_x n_y - 1$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
2	n_y+2	$\cdots$	$n_y(n_x-1)+2$
1	n_y+1	$\cdots$	$n_y(n_x-1)+1$
0	n_y	$\cdots$	$n_y(n_x-1)$

$$\text{index} = j + n_y i,$$

right layout

fast index on the right

Multidimensional array and data parallelism

Question: Assuming left layout, which loop would you prefer to parallelize (inner or outer) ?

row-major

$n_x(n_y-1)$	$n_x(n_y-1)+1$	$\cdots$	$n_x n_y - 1$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$2n_x$	$2n_x+1$	$\cdots$	$3n_x-1$
n_x	n_x+1	$\cdots$	$2n_x-1$
0	1	$\cdots$	n_x-1

index $= i + n_x j$,
left layout
fast index on the left

```
for(int j=0; j<ny; ++j)
  for(int i=0; i<nx; ++i)
    data[i+nx*j] += 12;
```

Favor memory locality:

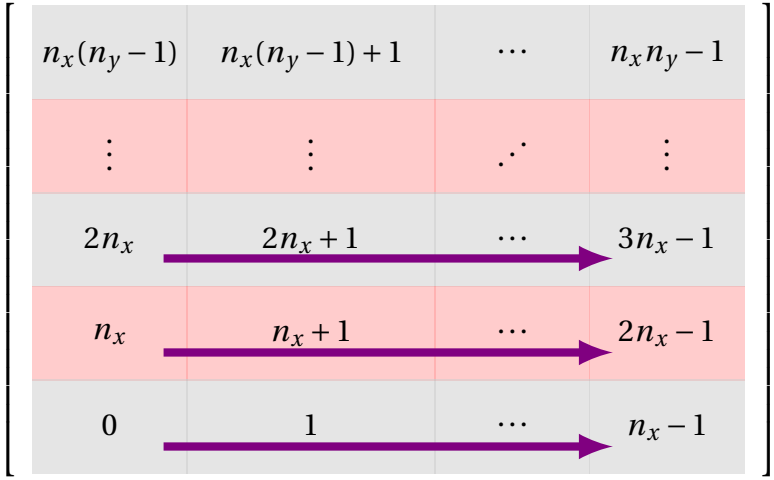
- maximize cache usage for CPU
- maximize memory coalescence on GPU

Different hardware ⇒ different parallelization strategies

Multidimensional array and data parallelism

Question: Assuming left layout, which loop would you prefer to parallelize (inner or outer) ?

row-major



$\text{index} = i + n_x j,$

left layout

fast index on the left

OpenMP // outer loop

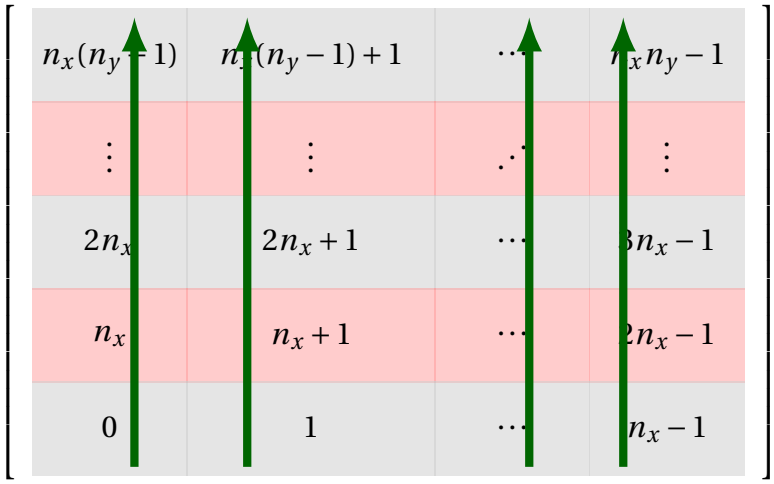
each OpenMP thread handles **1 or more row(s)**

```
#pragma omp parallel
{
  #pragma omp for
  for(int j=0; j<ny; ++j)

    // vectorization loop
    // memory contiguity
    for(int i=0; i<nx; ++i)
      data[i+nx*j] += 12;
}
```

Multidimensional array and data parallelism

Question: Assuming left layout, which loop would you prefer to parallelize (inner or outer) ?

row-major



index $= i + n_x j$, left layout
fast index on the left

CUDA // inner loop
each CUDA thread handles **1 or more col(s)**
memory coalescence

```
__global__ void compute(int *data)
{
  // adjacent memory cells
  // computed by
  // neighboring threads
  int i = threadIdx.x +
      blockIdx.x*blockDim.x;

  for(int j=0; j<ny; ++j)
    data[i+nx*j] += 12;
}
```

Multidimensional array and data parallelism

Conclusion:

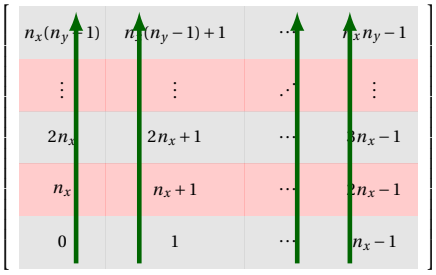
Don't assume layout, let's chose at compile-time !

- **First conclusion:**
  if we keep the same memory layout, **OpenMP** and **CUDA** **disagree** on which loop should be parallelized to optimize for their respective hardware target.
- **How can we make portable code ?**
- Note that swapping memory layout and `for loops` is **involutive**
- **Kokkos answer:** make memory layout abstract (since a good memory layout is hardware dependent), fixed at compile-time access $data(i, j)$
  - On **OpenMP** $data(i, j)$ actually means accessing $dataPtr[Ny * i + j]$
  - On **Cuda** $data(i, j)$ actually means accessing $dataPtr[i + Nx * j]$

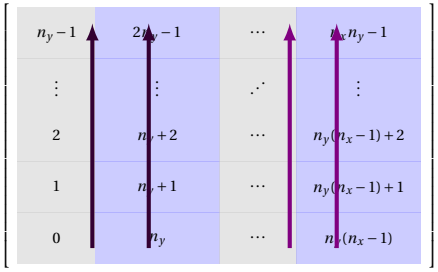
Multidimensional array and data parallelism

Don't assume layout, let's chose at compile-time !
Make it hardware aware.

left layout / CUDA



right layout / OpenMP



Kokkos parallel version for both CUDA/OpenMP

```
Kokkos::parallel_for(nx,
    KOKKOS_LAMBDA(int i) {
      for (int j=0; j<ny; ++j)
        data(i,j) += 12;
    }
);
```

7-point Stencil kernel with Kokkos - 1

- A **single high-level parallel programing model for shared memory** architectures (CPU, GPU, ...) ⇒ **developper more productive**
- 3d stencil kernel - **SERIAL**

```
// CPU version
for(int i=1; i<nx-1; ++i)
  for(int j=1; j<ny-1; ++j)
    for(int k=1; k<nz-1; ++k) {

      int index = k + j*nz + i*ny*nz

      y[index] = -5*x[index] +
        ( x[index-1]     + x[index+1] +
          x[index-nz]    + x[index+nz] +
          x[index-nz*ny] + x[index+nz*ny] );
    }
```

7-point Stencil kernel with Kokkos - 2

- A **single high-level parallel programing model for shared memory** architectures (CPU, GPU, ...) ⇒ **developper more productive**
- 3d stencil kernel - **parallel KOKKOS - naive**

```
// naive Kokkos kernel - for CPU, GPU, ...
Range3d range ( {{0,0,0}}, {{nx,ny,nz}} );

parallel_for(range, KOKKOS_LAMBDA(int i,
                                  int j,
                                  int k) {

  y(i,j,k) = -5*x(i,j,k) +
    ( x(i-1,j  ,k  ) + x(i+1,j  ,k  ) +
      x(i  ,j-1,k  ) + x(i  ,j+1,k  ) +
      x(i  ,j  ,k-1) + x(i  ,j  ,k+1) );
});
```

7-point Stencil kernel with Kokkos - 3

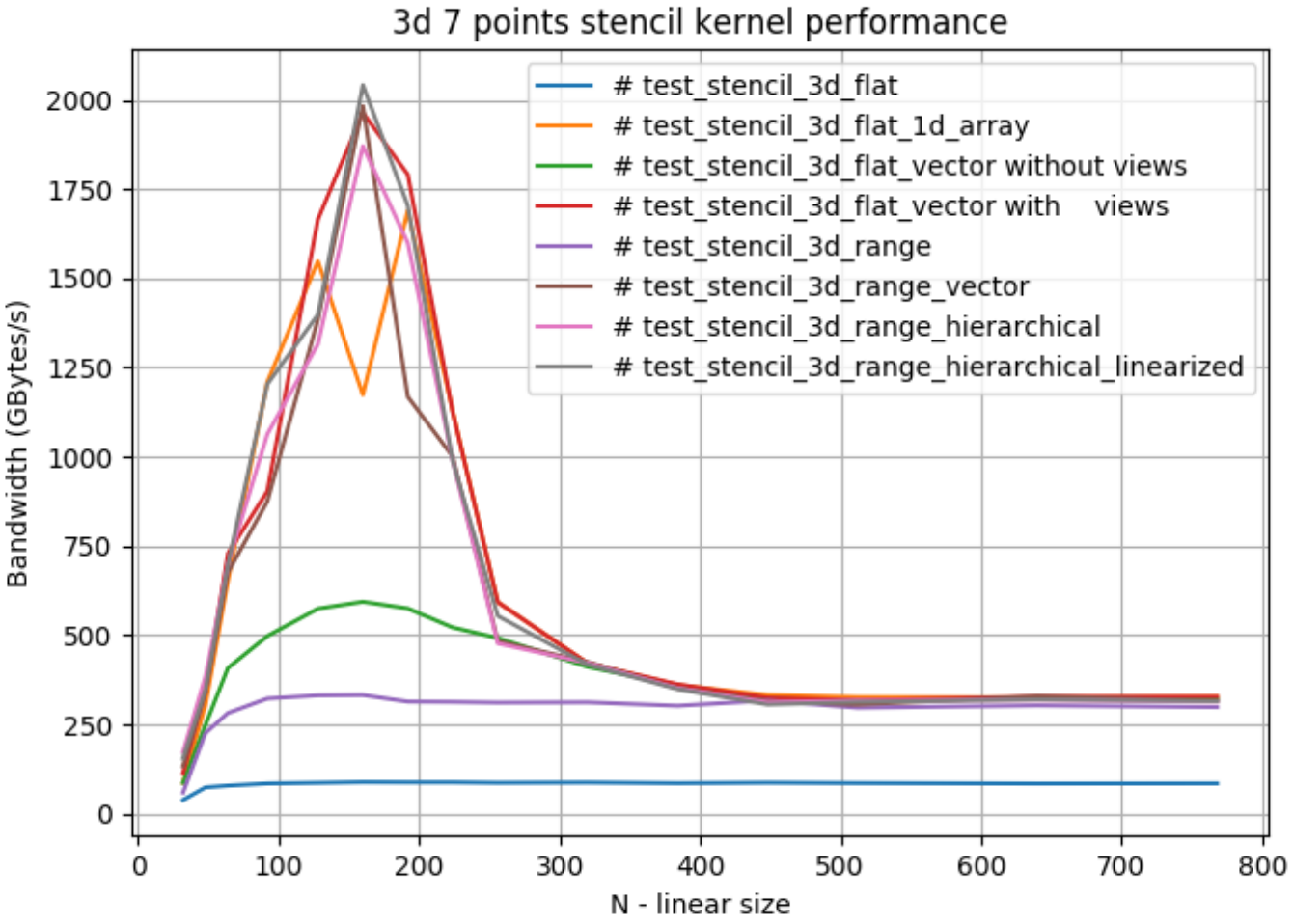
- A **single high-level parallel programing model for shared memory** architectures (CPU, GPU, ...) ⇒ **developper more productive**
- 3d stencil kernel - **parallel KOKKOS - HIERARCHICAL (CPU / GPU)**

```
parallel_for(team_policy_t(nbTeams, AUTO),
   KOKKOS_LAMBDA(const thread_t& thread) {
     int i = thread.league_rank();

     parallel_for(TeamThreadRange(thread, 1, n-1),
       [=](const int j) {

         parallel_for(ThreadVectorRange(thread, 1, n-1),
           [=](const int k) {

             if ( ... ) {
               y(i,j,k) = -5*x(i,j,k) +
               (x(i-1,j,k) + x(i+1,j,k) +
                x(i,j-1,k) + x(i,j+1,k) +
                x(i,j,k-1) + x(i,j,k+1));
             }
           }); // end vector range
       }); // end thread range
   }); // end team policy
```

ref: https://github.com/pkestene/kokkos-proj-tmpl

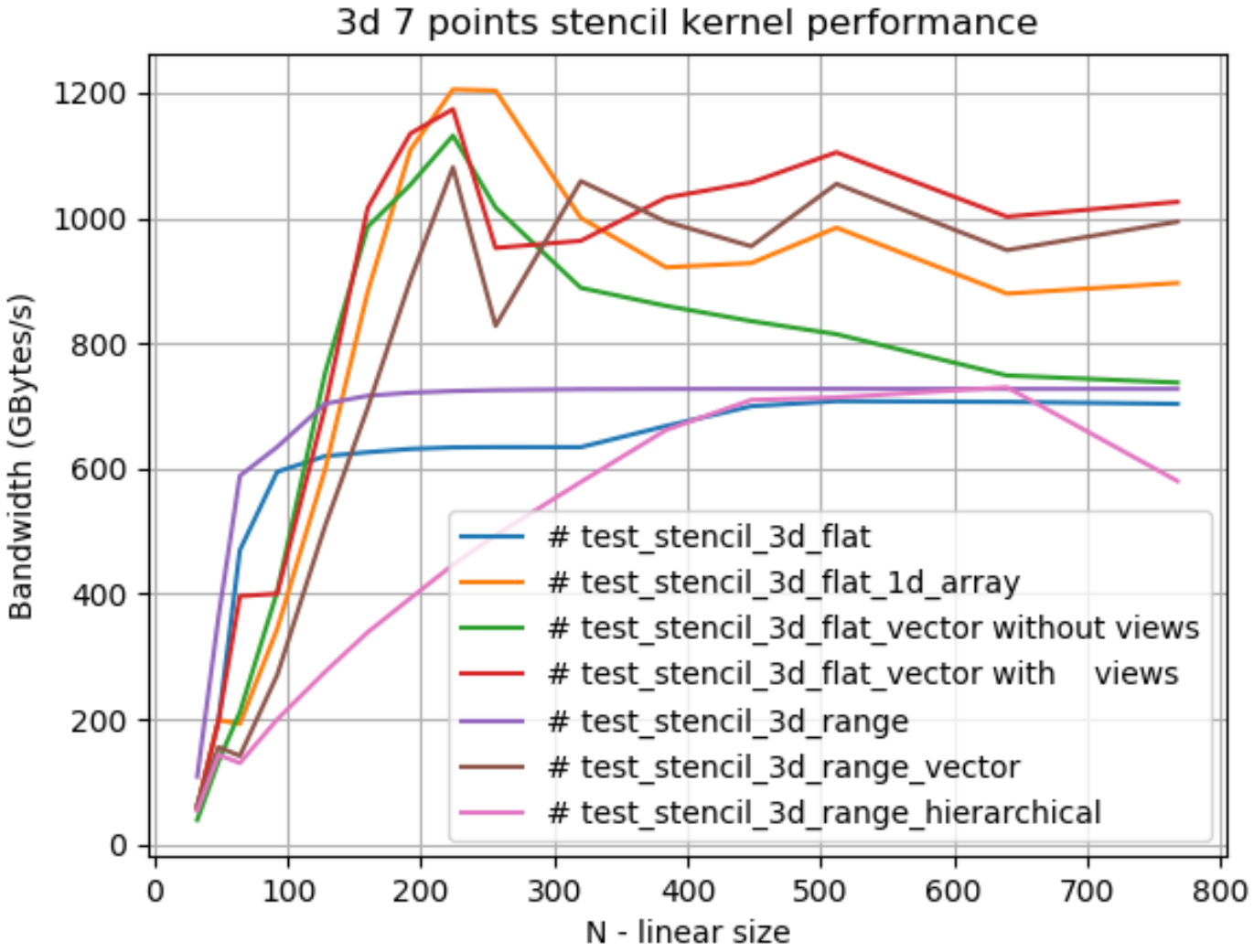
7-point Stencil kernel with Kokkos - 4

Intel CPU Skylake 2×18 cores, gold



7-point Stencil kernel with Kokkos - 4

Nvidia GPU P100, ouessant@IDRIS



List of Kokkos related activities at MDLS

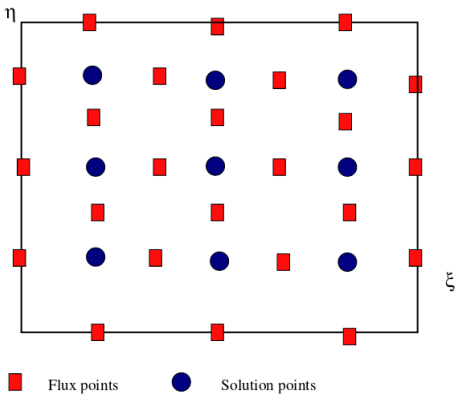
- `EulerKokkos:`, large miniApp, prototype hydro/MHD 2D/3D (2nd order Finite Volume MHD), MPI+Kokkos, spawned skeleton, ~ 20 kSLOC
- `ARK:` derived `EulerKokkos` with new numerical scheme for all-Mach number flow + radiative transfert (TP+HB), under ERC grant project ATMO (P. Tremblin); also multiphase-flow (WIP, very recently)
- `ppkMHD:` extended with high-order hydrodynamics (SDM, Mood, ...) MPI+Kokkos, prototyping for solar physics, ERC WholeSun (PK, A.S. Brun); ~ 30 kSLOC
- `LBM_Saclay:` multiphase flow (NavierStokes + Conservative Phase-field model+Thermics), prototype code, PhD thesis starting in September 2019; ~ 9 kSLOC

Code ppkMHD: high-order hydrodynamics

High-order SDM (Spectral Difference Methods)

- **Euler conservation law :** $\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z} + M = 0$
- SDM implementation up to order $N = 6$
- N^d **solution (DoF) points**
  (Gauss-Chebyshev): $x_s = \frac{1}{2}\left[1 - \cos\left(\frac{2s-1}{2N}\pi\right)\right]$
- $N^{d-1}(N+1)$ **flux points per direction**
  (Gauss-Legendre): use the roots of Legendre polynomial of degree $N-1$ + the two end points
- Use **1D (tensor product) Lagrange polynomials** to represent solution.

reference:
Spectral difference method for compressible flow on unstructured grid mixed elements, Liang et al, JCP, vol 228, 2009



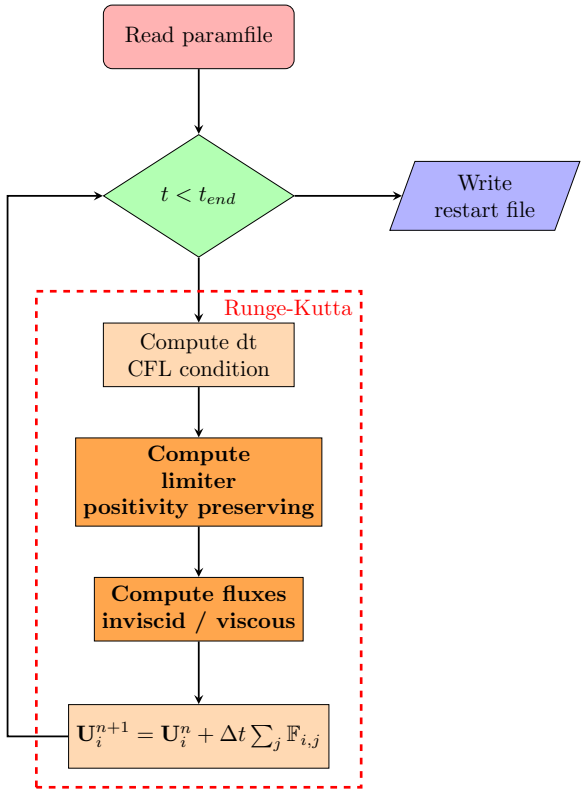
- Lagrange interpolation from **solution points** to **flux points** (and opposite flux to solution)
- Interpolation operators `sol2flux` and `flux2sol` are implemented via (small size) matrix-vector multiplication

Code ppkMHD

High-order SDM (Spectral Difference Methods) ingredients

https://gitlab.maisondelasimulation.fr/pkestene/ppkMHD

(avail upon request)



- **MPI + Kokkos parallelization (Intel CPU, Nvidia GPU, ARM CPU, ...)**
- **SSP (strong stability preserving) Runge-Kutta**
- **No articial viscosity for stability.**
- **TVD limiter:**

  A Spectral Difference Method for the Euler and Navier-Stokes Equations on Unstructured Meshes, by Wang et al., J. Sci. Comp., 2007
- **Positivity preserving:** adapt ideas from DG to SDM

  On positivity-preserving high order discontinuous Galerkin schemes for compressible Euler equations on rectangular meshes, Zhang et al, JCP 2010, vol. 229, Issue 23.

Kokkos implementation of SDM schemes

example functor: interpolate at flux points

```
// One thread per cell.
template<int dim, int N, int dir>
class Interpolate_At_FluxPoints_Functor : public SDMBaseFunctor<dim,N> {
  // ...
  //! functor for 3d
  KOKKOS_INLINE_FUNCTION
  void operator()(const size_t& index) const {

    for (int idz=0; idz<N; ++idz) {
      for (int idy=0; idy<N; ++idy) {
        // for each variables
        for (int ivar = 0; ivar<nbvar; ++ivar) {
          // get solution values vector along X direction
          for (int idx=0; idx<N; ++idx)
            sol[idx] = UdataSol(i,j,k, dofMapS(idx,idy,idz,ivar));
          // interpolate
          this->sol2flux_vector(sol, flux);
          //then write results in output buffer
          // ....
        } // end for ivar
      } // end for idy
    } // end for idz
  } // end operator()
};
```

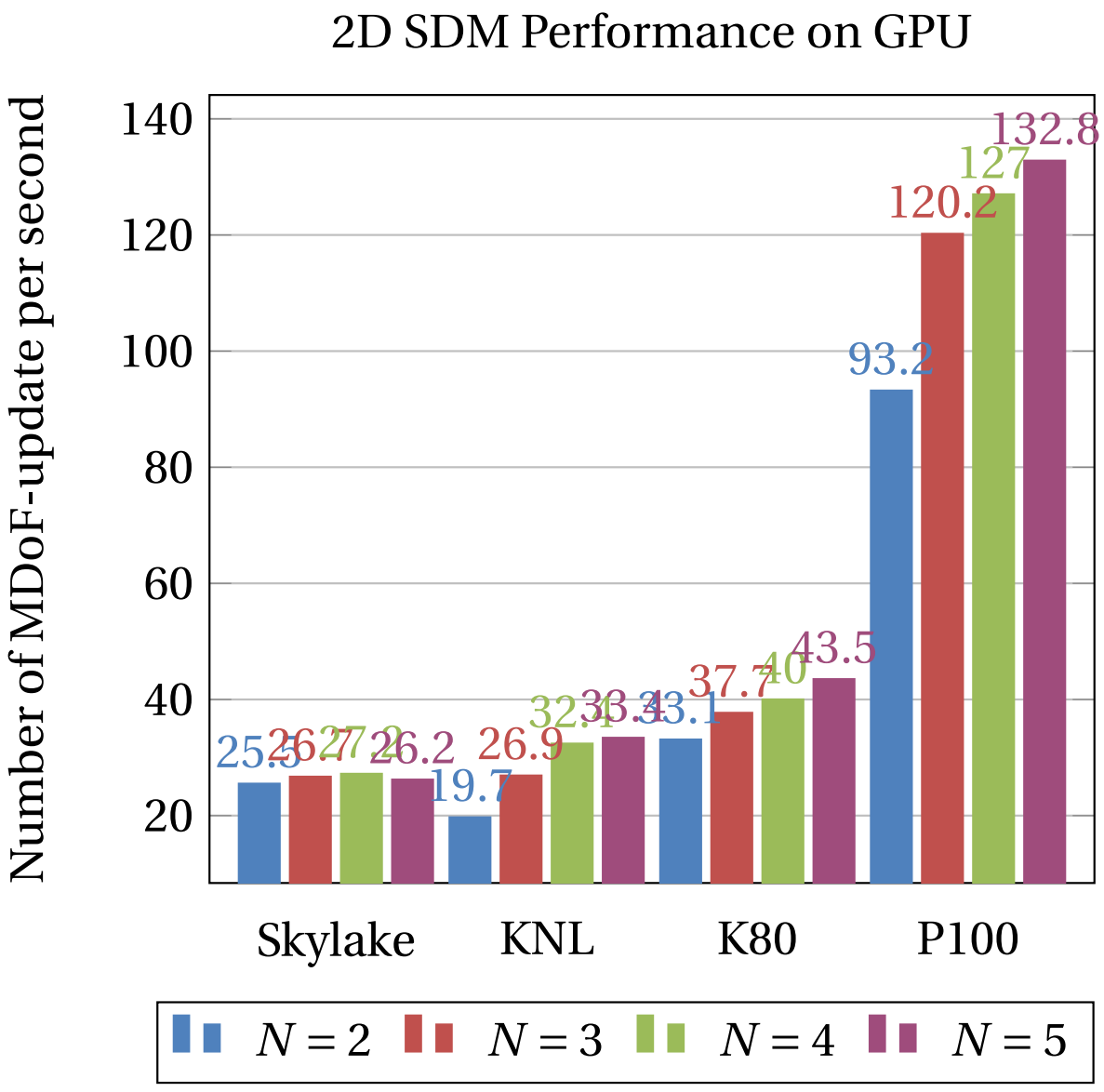
High-order numerical scheme comparison - SDM

- **SDM degree 2** vs **SDM degree 4** for Compressible Euler, TVD_RK3
- **same # DoFs :** 400^2 **degree 2** $\Leftrightarrow$ 200^2 **degree 4**
- ⚠ **high-order** $\Rightarrow$ **CFL constraint more restrictive**
- Time to solution (1 GPU, Pascal P100):

nb cells	#DoFs	degree	time(seconds)	speed (Dofs/s)
SDM 200^2	400^2	2	5	57
SDM 200^2	800^2	4	25	101
SDM 400^2	800^2	2	23	93
SDM 400^2	1600^2	4	156	127
SDM 800^2	1600^2	2	155	111
SDM 800^2	3200^2	4	1150	138

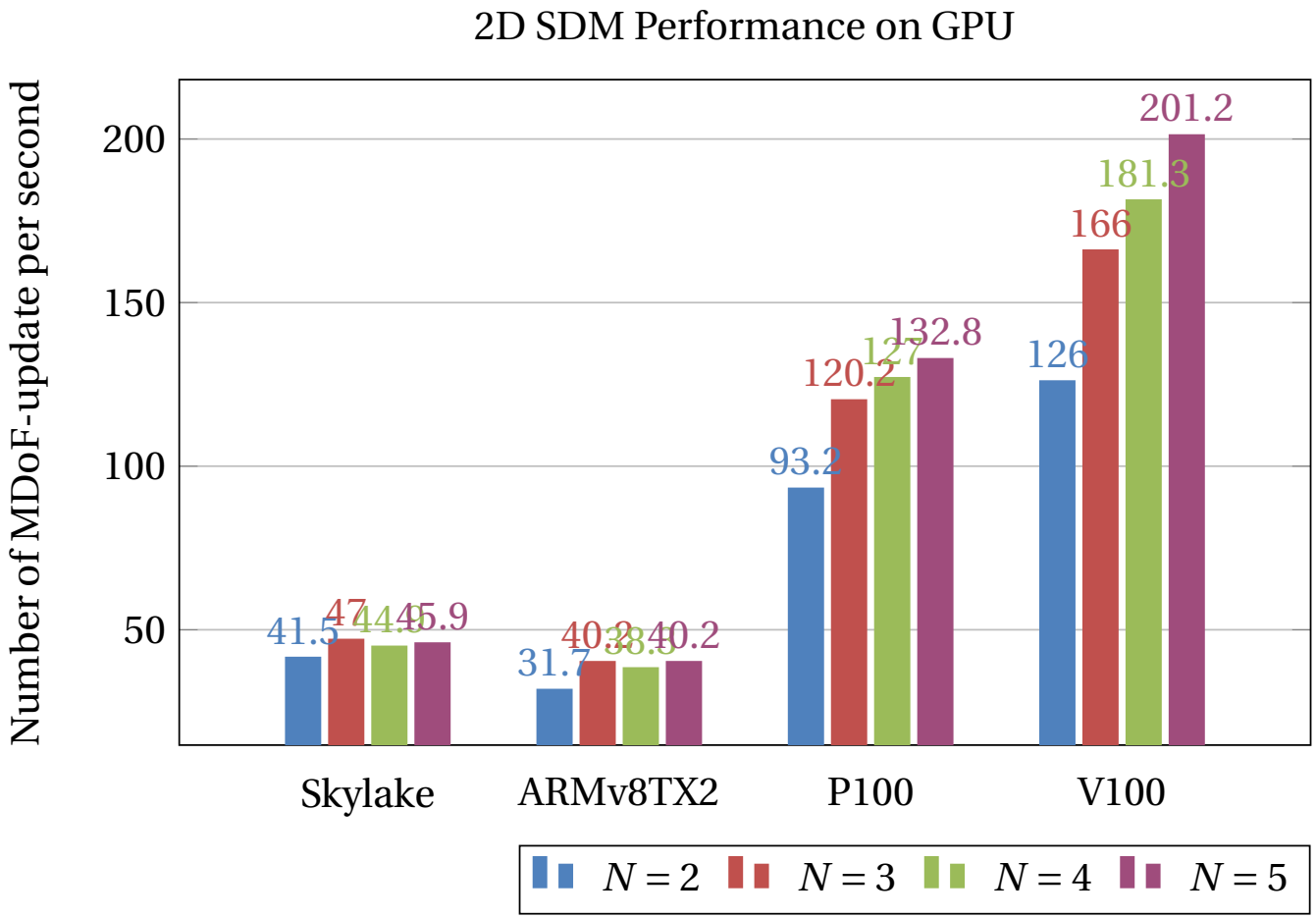
- **SDM implementation is more efficient for high degree** (ratio compute/bandwidth higher $\Rightarrow$ better for GPU)

2D SDM schemes - Intel Skylake vs Nvidia P100



- Test on skylake performed on `alfven` at **CEA/IRFU**.
- KNL test on PRACE Irene
- Skylake compiler is **INTEL icpc 18.0**
- Time integration is RK3
- **Pascal P100** is $\sim x2.5$ faster than **Skylake** (20 cores - dual socket - 2018)
- 2018 Skylake performs better than Nvidia K80

2D SDM - Intel Skylake vs ARM TX2 vs Nvidia P100



Quantitavely assessing performance portability

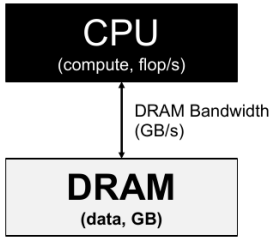
How to measure performance portability ?

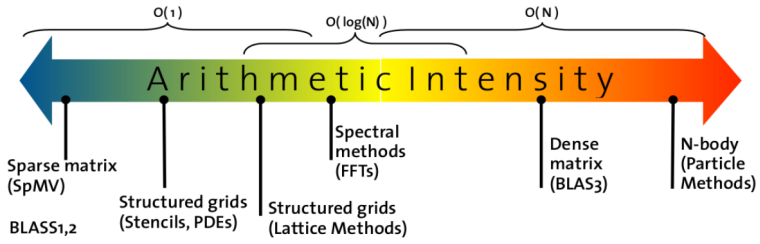
- **Application efficiency:** ratio of code's performance on a given architecture to that of the its best implementation on all architecture
- **Architectural efficiency:** ratio of the code's actual performance to that architecture's peak performance
  $\Rightarrow$ roofline model
- Prefer benchmark rather then vendor's own specific number
- Counting FLOPS must be done carefully (FMA, divide, exponential, ...) are very architecture dependant
- Memory access pattern may impact vectorization
- sustained performance

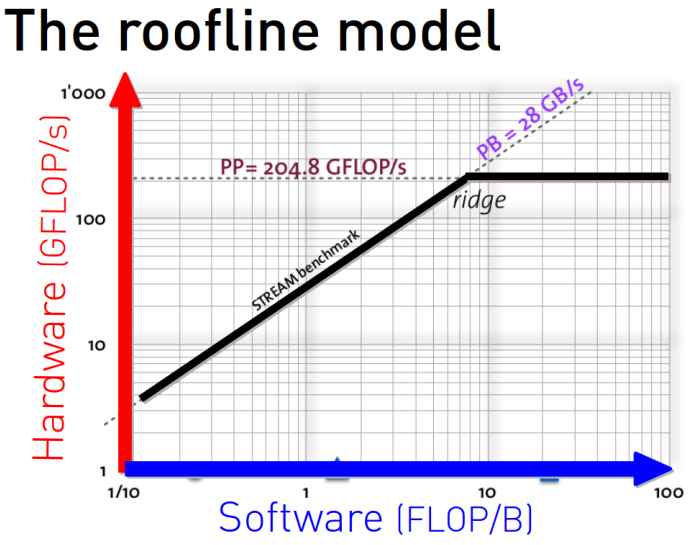
reference : An Empirical Roofline Methodology forQuantitatively Assessing Performance Portability, 2018, IEEE/ACM Int. Workshop of Performance, Portability and Productivity in HPC.

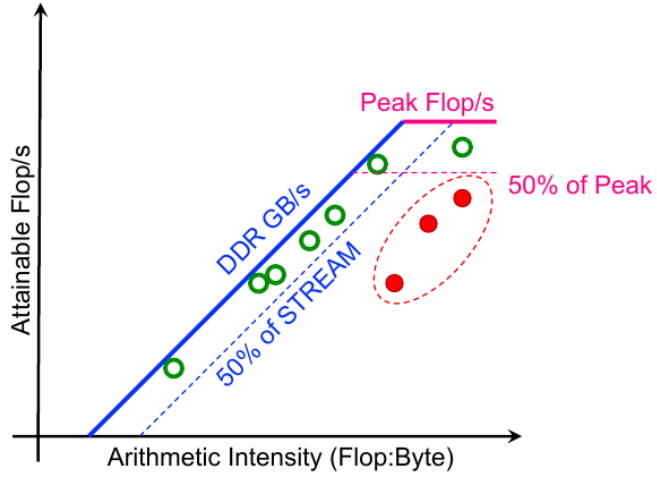
Roofline model

- What is it ? A way to analyze, understand, access the performance of an algorithm implementation
- **GFlops** versus **arithmetic intensity**
- Intel advisor can automate measurements



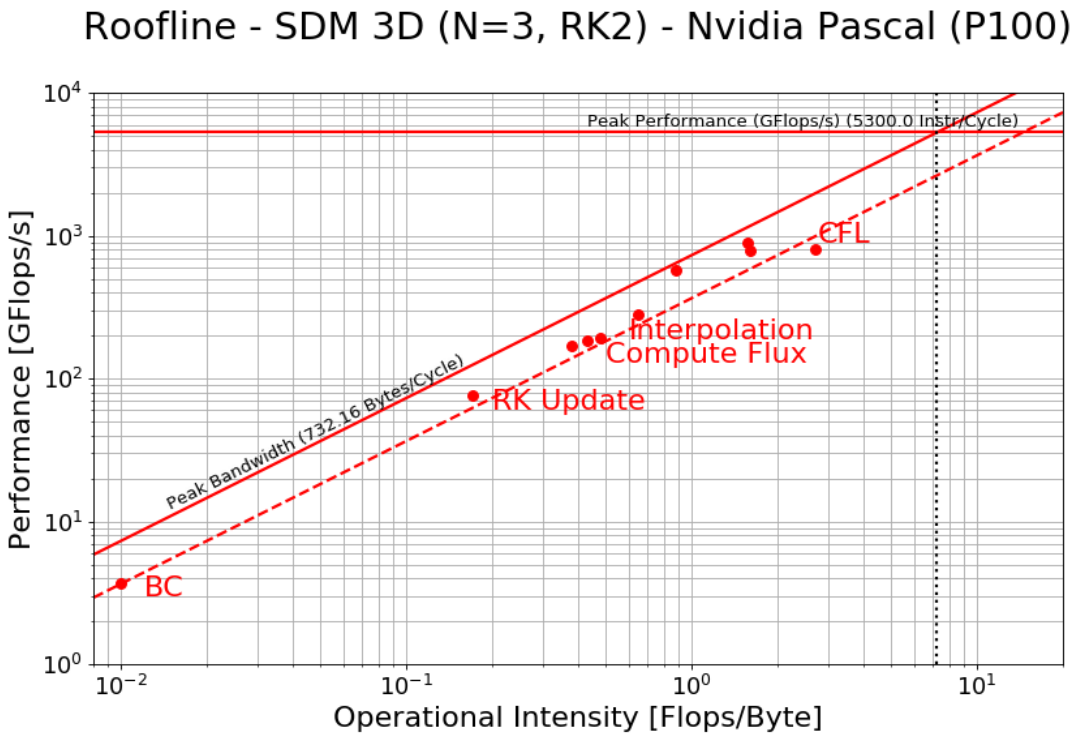






Spectral difference methods solver - Implementation

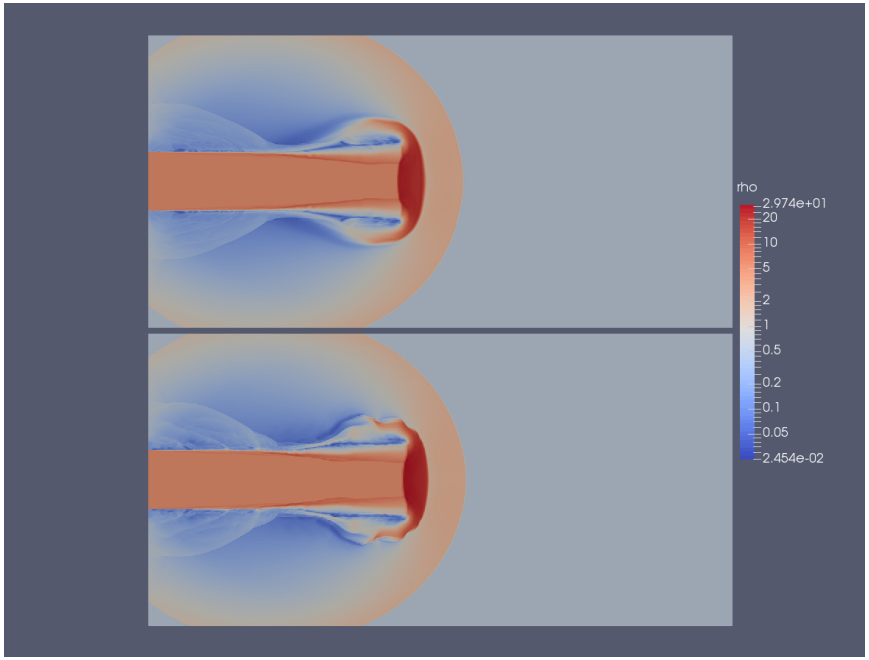
current SDM implementation - bandwidth limited !



- dashed line is 50% of peak bandwidth
- performed @ouessant (IDRIS/GENCI, IBM POWER8 + NVIDIA P100)

Spectral difference methods - Jet test - High Mach flow

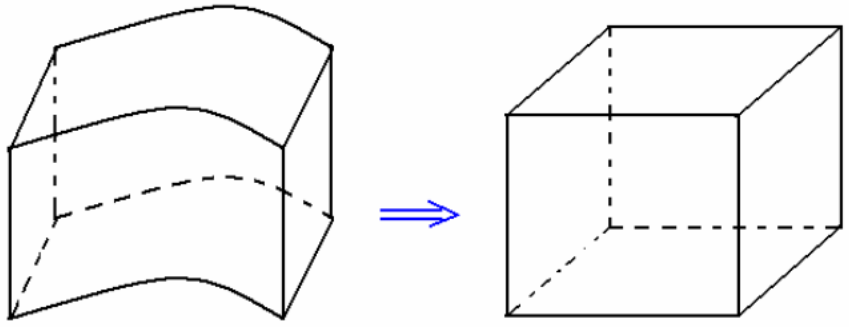
SDM scheme, Mach=27, comparison between order 3 and 4



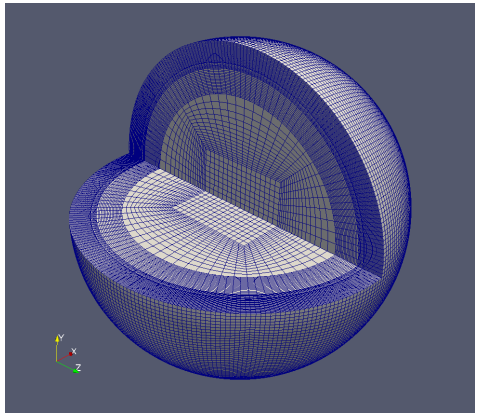
Code ppkMHD

- **Futur developments: towards multi-architecture AMR with Kokkos**
- **adapt SDM scheme to spherical geometry via coordinate transformation + mesh refinement** (CanoP)

$$\partial_t Q + \partial_x F + \partial_y G + \partial_z H + M = 0 \Rightarrow \partial_t \tilde{Q} + \partial_\xi \tilde{F} + \partial_\eta \tilde{G} + \partial_\zeta \tilde{H} + \tilde{M} = 0$$



- adapt SDM schemes for MHD
- **towards global solar dynamo and surface physics with Sacha BRUN (CEA)**



Kokkos for Fortran users ?

See DOE performance portability meeting, 2019, slides by Geof Womeldorff (LANL), `Methods for Fortran / Kokkos interoperability`.

- Keep main code drivers in fortran, but hotspots kernel in C++/Kokkos
- **incremental porting for large legacy codes**
  $\Rightarrow$ start with data allocated on the fortran side, focus on designing Kokkos computational kernels (this will be a functional porting for host), then use `Kokkos::DualView` to manage mirroring host and device memory data.

Who uses Kokkos ?

Kokkos Based Projects

Sandia National Laboratories

- 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 **30**
- Packages In Large Collections (e.g. Tpetra, MueLu in Trilinos) committed to using Kokkos and actively porting
  - Somewhere around **50**
- Counting also proxy-apps and projects which are evaluating Kokkos (e.g. projects who attended boot camps and trainings).
  - Estimate **80-120** packages.

source: C. Trott, Kokkos PI, DOE PPP Meeting, April 2019

Who uses Kokkos ?

source: C. Trott, Kokkos PI, DOE PPP Meeting, April 2019

The future of portable asynchonous tasking

HiHAT initiative: Hierarchical Heterogenous Asynchronous Tasking

- for Runtime frameworks developpers + HW vendors
- Tasking frameworks

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/

The future of portable asynchonous tasking

HiHAT initiative: Hierarchical Heterogenous Asynchronous Tasking

- for Runtime frameworks developpers + HW vendors
- 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)
- 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

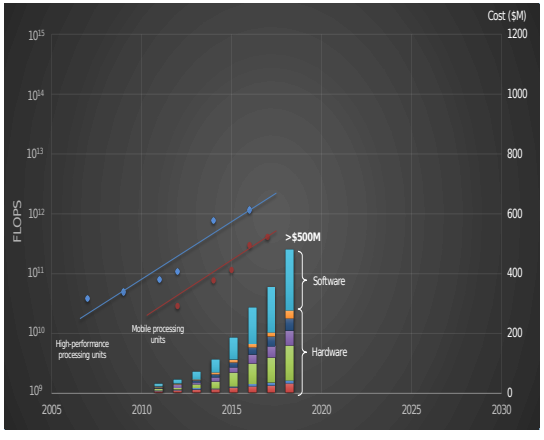
others runtime systems:

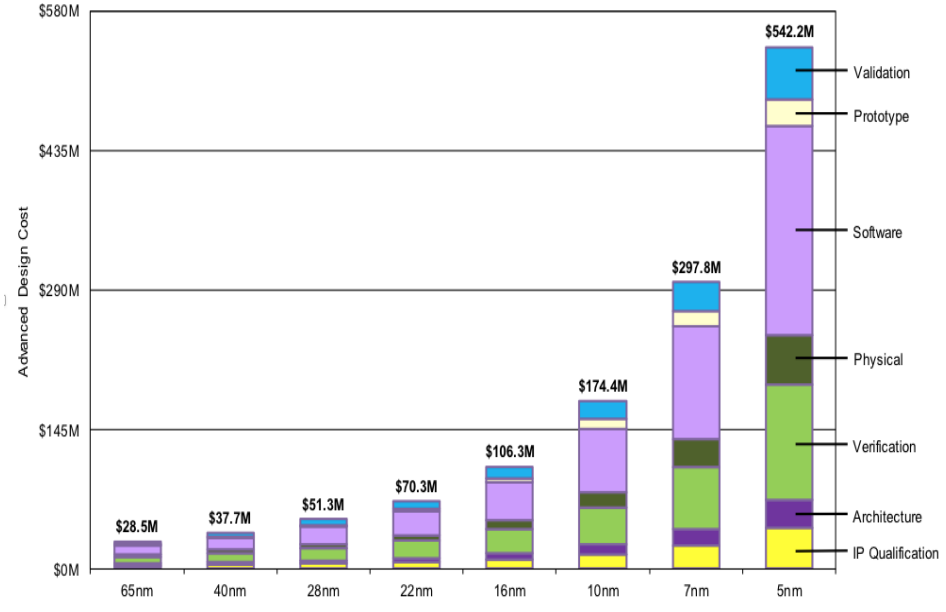
`https://hihat-wiki.modelado.org/Runtime_Clients`

Comparative analysis of Legion, Charm++, Uintah

Beyond Moore's Law ...

- Semiconductor sector still growes faster than GDP (PIB)
- Current chip makers (Intel, NVidia, Xilinx, ...) increasingly focus on specific applications : AI,... not necessarily HPC driven anymore
- **Skyrocketing cost of chip development limits hardware innovation.**
- June 2017 : **DARPA Electronics Resurgence Initiative** : 5-year 1.5 B$ investment for new chip architectures, IC design and materials and integration. ⇒ **reduce costs of chip design**

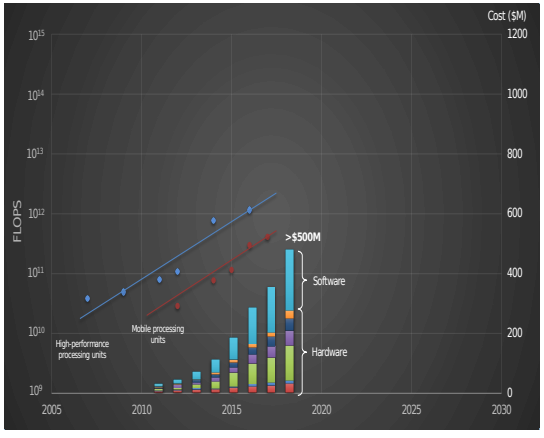


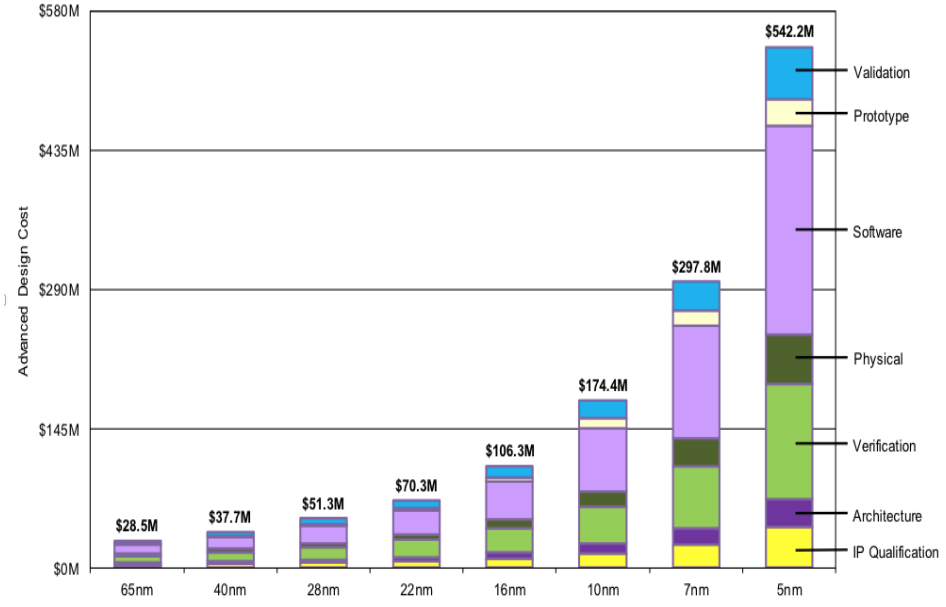


https://www.enterprisetech.com/2018/07/24/darpa-effort-pushes-beyond-moores-law

Beyond Moore's Law ...

- **The cost of chip making... needs huge revenue to justify these expenses**
- Nvidia Volta/Turing (12 nm), next-gen in 7 nm
- AMD Vega / ZEN2 (2019): 7nm
- Xilinx Versal FPGA (2019): 7nm
- Fujitsu ARM 64 bit for HPC (2019-2020): 7nm
- **nothing beyond 3nm (if reachable) ? ⇒ 3D integration package**





https://www.enterprisetech.com/2018/07/24/darpa-effort-pushes-beyond-moores-law

Beyond Moore's Law ...

- **What's next ?**
- **Rethinking hardware architecture design:**
  - not necessarily increasing die size
  - 2.5D or 3D packages integration
  - Open source hardware : e.g. RISC-V (young tech., 2015, just a co-processor ?)
    ⇒ good for startup innovation, prototyping, many hardware *flavors*, on-chip heterogenous computing
    ⇒ fast growing markets (AI / IoT, automotive)
  - many different end markets: IoT, automotive, data centers (cloud, AI)...
  - **More hybrid architectures ?** see e.g. **Xilinx versal (2019) all-in-one**