Implementing Spectral Difference Methods (SDM) for Compressible Euler flow simulations using performance portable library kokkos

Pierre Kestener, Sacha Brun

CEA Saclay, DRF, Maison de la Simulation, FRANCE

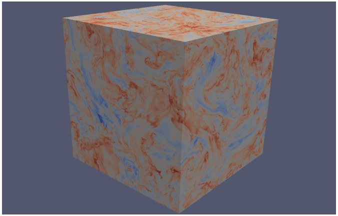
Astrosim conference, Lyon, October 9th, 2018

Content

- **Motivations:** computational sciences and software engineering
- **Short Kokkos overview**: a C++ library for performance portability, **a new way of designing portable parallel codes**
  - Refactoring or designing Hydrodynamics / MHD kernels
  - Same performance between old CUDA kernels and new Kokkos Kernels?
- **Implementing SDM high-order numerical schemes with Kokkos**
- **Performance measurements on multiple architectures:**
  - Intel Skylake, ARM ThunderX2 : device Kokkos::OpenMP
  - Nvidia GPU Pascal P100 : device Kokkos::Cuda
- **Perpectives / Future applications and developments**
  - **SDM Integration into our AMR code** `CanoP`

Motivations of this work - 1

- code RAMSES-GPU : **Magneto-Rotational Instability, MHD turbulence, ...**
  - developped in CUDA/C++ for **astrophysics applications** on regular grid
  - $\sim$ 70k lines of code (out of which $\sim$ 16k in CUDA)
  - developed between **2009 and 2014** !
- Since then **both GPU hardware/sofware have tremendously evolved** (in orders of magnitude in memory bandwidth, number of registers per SM, c++11, ...) $\Rightarrow$ a lot of optimization techniques accumulated over the years are **not so critically important anymore** on today's GPU.
- Collaborations with domain scientists are hard when required software skills include CUDA.
- **2016-2017** is the right time to refactor code, **sparkle new ways to develop scientific software at a higher abstraction level**
- **Can we rewrite an application like RamsesGPU in a new high-level approach for better software/science productivity ?**

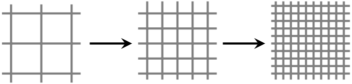


Motivations of this work - 2

- **Software engineering**
  - Refactoring existing C++/CUDA code
  - As much as possible **performance portable code:** write the code once, and let the user run it on the available target platform with performance as good as possible.
  - Prefer a **high-level approach** among:
    - **Directive-based:** OpenACC, OpenMP
      ease of use, incremental approach, **for large legacy code bases**, ...
    - **External *smart* library** implementing **parallel programming patterns** (for, reduce, scan, ...):
      Kokkos, RAJA, agency, arrayFire libraries are such possibilities
      parallel programing patterns as 1^{st} class concepts, architecture adapted data containers, c++ integration / engineering, ...
    - **Other high-level approaches (more experimental):** SYCL (Khronos Group *standard*), hpx (heavy use of new c++ standards (11,14,17): `std::future`, `std::launch::async`, distributed parallelism, ...)

Motivations of this work - 3

- **Computationnal science ground - Computational Fluid Dynamics**
  - **High-order numerical schemes** for **compressible hydrodynamics**
  - **How fast the numerical solution converges to the *reference* solution when increase space resolution ?** $|f - f_r| \leq h^{-N}$



  - From a discussion with Sacha Brun @ CEA, DAp, *A compressible high-order unstructured spectral difference code for stratified convection in rotating spherical shells* by Wiang, Liang and Miesch, JCP 2015
  - **Spectral Difference Methods** is a **high order scheme familly** $\simeq$ **Discontinuous Galerkin**
    - **Spectral Difference Methods** have simpler formulation, (should be) more efficient (esp. high order)
    - **Discontinuous Galerkin**, more accurate

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 do I mean by 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):

$$index = i + nx * j$$

- **Fortran** (column-major format) vs **C/C++** (row-major format) ⇒ **memory layout should be hardware-aware configurable**
- 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)
- 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**)

7-point Heat kernel with Kokkos - 1

- A **single high-level parallel programing model for shared memory** architectures (CPU, GPU, ...) ⇒ **developper more productive**
- 3d heat (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 Heat kernel with Kokkos - 2

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

```
// 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 Heat kernel with Kokkos - 3

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

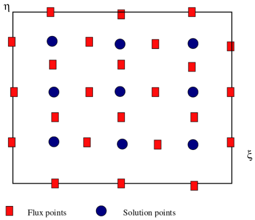
```
// Kokkos kernel to promote compiler vectorization, e.g. for Intel Skylake
Range2d range ( {{0,0}}, {{nx,ny}} );

// only parallelize the 2 outer loops (i and j)
parallel_for(range, KOKKOS_LAMBDA(int i,
                                  int j) {

  // create 1d subview along z axis - same as a 1d slice in fortran
  auto xij = subview(x,i,j,Kokkos::ALL());
  auto ...

  // only use 1d slices
  // let the compiler vectorize the k-loop
  for (int k=1; k<nz-1; ++k)
    yij(k) = -5*xij(k) +
      ( xij_1(k) + xij_2(k) +
        xij_3(k) + xij_4(k) +
        xij(k-1) + xij(k+1) );
});
```

Spectral difference methods solver - SDM

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

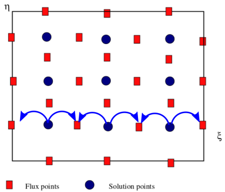
Spectral difference methods solver - SDM

High-order SDM (Spectral Difference Method)

- **Euler conservation law :**
  $$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z} + M = 0$$
- Use **1D (tensor product) Lagrange polynomials** to represent solution:

$$Q(x,y) = \sum_{i=0}^{i=N-1} \sum_{j=0}^{j=N-1} Q_{i,j} l_i(x) l_j(y)$$

where l_i is the Lagrange polynomial such that $l_i(x_j) = \delta_{i,j}$ and $\delta_{i,j}$ are solution point locations.



- **step1:** Lagrange interpolation from **solution points** to **flux points**

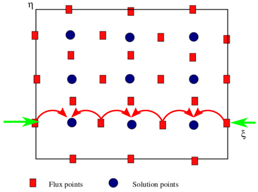
Spectral difference methods solver - SDM

High-order SDM (Spectral Difference Method)

- **Euler conservation law :**
  $\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z} + M = 0$
- Use **1D (tensor product) Lagrange polynomials** to represent solution:

$$Q(x,y) = \sum_{i=0}^{i=N-1} \sum_{j=0}^{j=N-1} Q_{i,j} l_i(x) l_j(y)$$

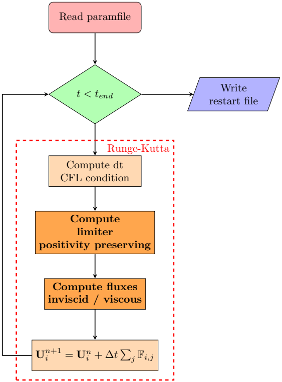
where l_i is the Lagrange polynomial such that $l_i(x_j) = \delta_{i,j}$ and $\delta_{i,j}$ are solution point locations.



- **step2:**
  - solve Riemann problem at end points
  - evaluate fluxes at **flux points**
  - interpolate fluxes at **solution points**

Spectral difference methods solver - SDM

High-order SDM (Spectral Difference Methods) ingredients



- **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.

High-order numerical scheme comparison - SDM

SDM degree 2 - device `Kokkos::Cuda`

SDM degree 4 - device `Kokkos::Cuda`

resolution: 200^2, 400^2, 800^2

Performed on system `ouessant` (Nvidia GPU P100) at **IDRIS/GENCI**, France.

High-order numerical scheme comparison - SDM

- **SDM degree 2** vs **SDM degree 4** for Compressible Euler, TVD_RK3
- **same # DoFs :** 400^2 **degree 2** ⇔ 200^2 **degree 4**
- ⚠ **high-order** ⇒ **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 ⇒ better for GPU)

Spectral Difference Method convergence

- Use the **isentropic vortex advection** test **(exact solution of compressible Euler flow)**: periodic boundary conditions, vortex should returns to the initial conditions at $t = 10.0$

$$T = T_0 - \frac{(\gamma - 1) * \beta^2}{8\gamma\pi^2} e^{1-r^2}$$

$$\rho = \rho_0 \frac{T}{T_0}^{\frac{1.0}{\gamma-1}}$$

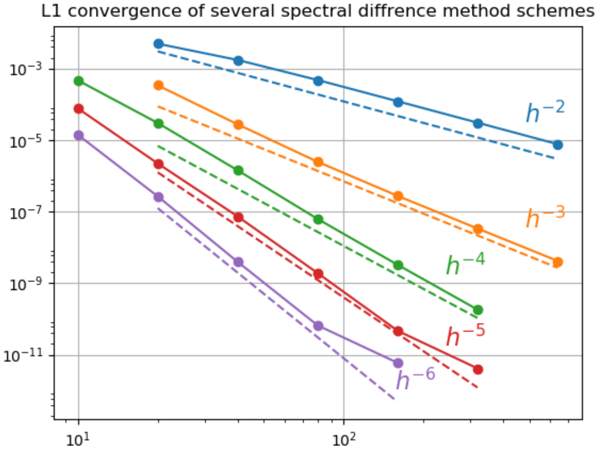
$$\rho u = \rho \left(u_0 - (y - y_0) \frac{\beta}{2\pi} e^{0.5*(1.0-r^2)} \right)$$

$$\rho v = \rho \left(v_0 + (x - x_0) \frac{\beta}{2\pi} e^{0.5*(1.0-r^2)} \right)$$

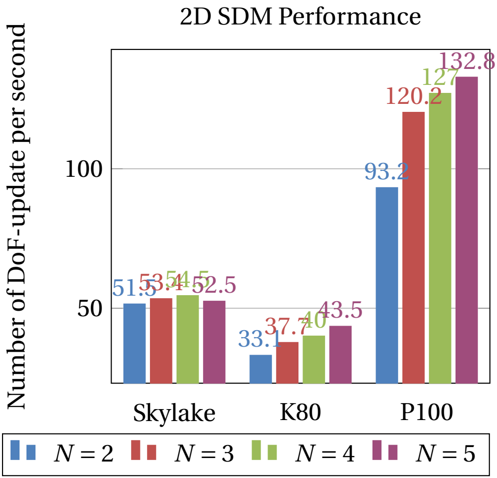
$$\rho e = \frac{\rho T}{\gamma - 1} + \frac{1}{2} \rho (u^2 + v^2)$$

reference:
https://www.cfd-online.com/Wiki/2-D_vortex_in_isentropic_flow

Spectral Difference Method convergence

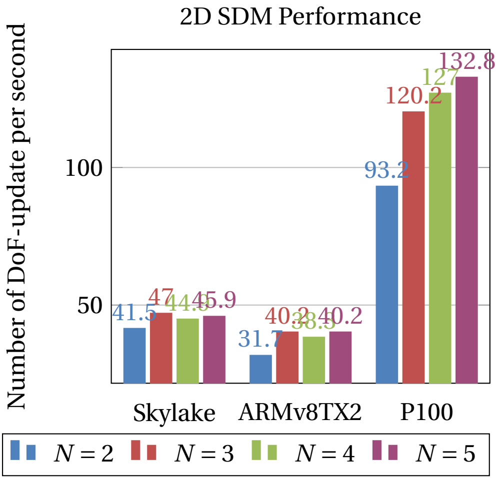


2D SDM schemes - Intel Skylake vs Nvidia P100



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

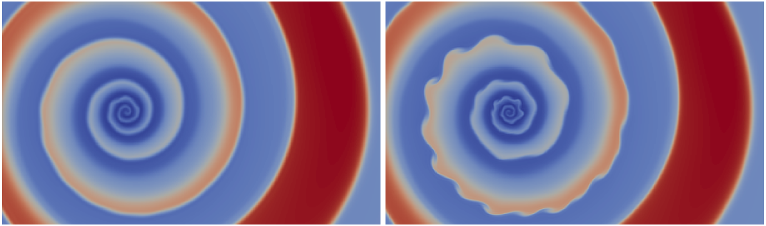
2D SDM - Intel Skylake vs ARM TX2 vs Nvidia P100



- Test on skylake (dual socket) performed on `alfven` at **CEA/IRFU**.
- Test on ARMv8TX2 (dual socket) performed on **GENCI prototype @ CEA/DAM**
- Test on P100 performed on **GENCI prototype** `ouessant` **@ IDRIS**
- Skylake compiler is **GNU g++ 7.3**
- ARMv8TX2 compiler is **GNU g++ 7.1**
- Time integration is RK3

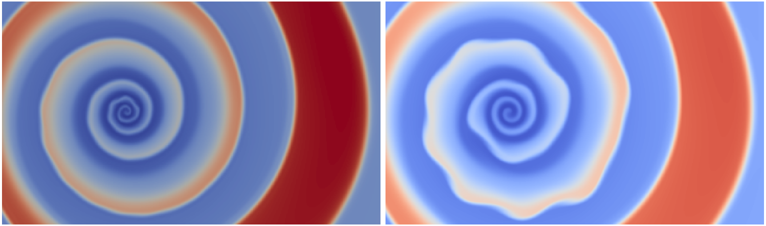
Spectral difference methods - numerical viscosity

- **Effet of numerical viscosity:** illustration using same number of #Dof for the Kelvin-Helmholtz setup:
  - SDM, degree3, 512^2
  - SDM, degree6, 256^2



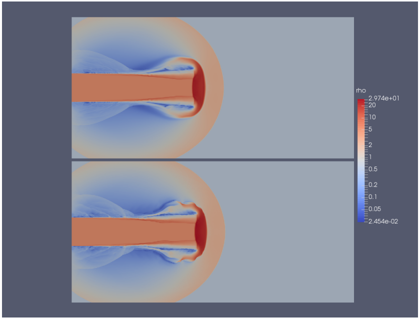
Spectral difference methods - numerical viscosity

- **Effet of numerical viscosity:** illustration using same number of #Dof for the Kelvin-Helmholtz setup:
  - SDM, degree3, 512^2
  - MUSCL, Finite Volume, degree2, 1536^2



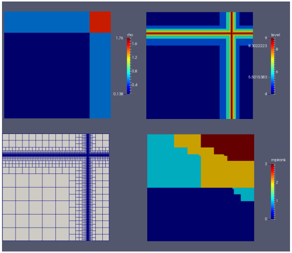
Spectral difference methods - Jet test - High Mach flow

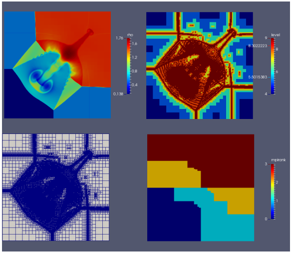
SDM scheme, Mach=27, comparison between order 3 and 4



CanoP - a parallel adaptive mesh refinement framework

- **What is** `CanoP` **?** **An applicative layer on top of** `p4est` **(distributed mesh management library)**
- `CanoP` **wraps the core** `p4est` **functionalities** in a set of a few C++ class
- `CanoP` **provides a template application framework:** new users don't need to have a deep knowledge of how `p4est` works
  - parallel IO (HDF5+XDMF),
  - input parameter file management (LUA),
  - Init, border conditon factory,
  - refine/coarsen indicator factory



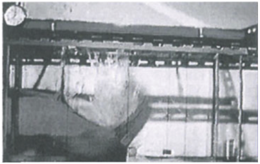


List of solvers available in CanoP

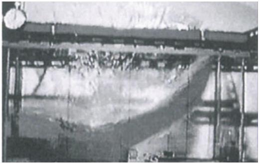
- **Some pedagogical schemes** (for training new users):
  - finite volume scalar advection, **A. Fikl**
  - scalar viscous/invicid Burgers equation, **Q. Wargnier, R. DiBattista, PK**
- **bifluid**: a two-phase flow model (**F. Drui, A. Fikl**, A. Larat; S. Kokh, M. Massot)
- **ramses**: monophasic Euler with 2nd order MUSCL-Hancock numerical scheme, for astrophysics applications, **PK**, poisson solver (O. Iffrig, PK), adaptive time stepping (**O. Iffrig**)
  - study angular momentum transport in accretion disk: **N. Brucy / W. Verdier** M1 intership, 2018, P. Hennebelle, O. Iffrig, PK)
- **Spray**: droplet evaporation modeling with a kinetic approach, **M. Essadki, PhD thesis**, M. Massot, S. De Chaisemartin
- **BN**: two-phase flow with Baer-Nunziato model (**F. Chen, PhD student**, A. Allou, JD Parisse, S. Kokh, PK)
- **MHD-KT**: WIP - magneto-hydrodynamics (MHD) with Kurganov-Tadmor discretization, multi-component plasma, solar physics, magnetic reconnection problem (**Q. Wargnier, PhD student**, M. Massot, PK)
- **ramsesRT**: WIP - Euler equations with radiative transfer (**H. Bloch, PhD student**, MDLS, 2018, P. Tremblin, M. Gonzalez, A. Audit)

CanoP : Two-phase flow solver

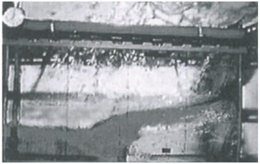
Experiment:



0.2s



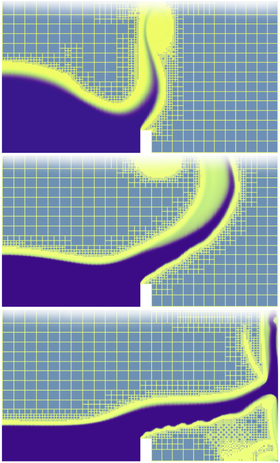
0.3s



0.4s

Credit F. Golay

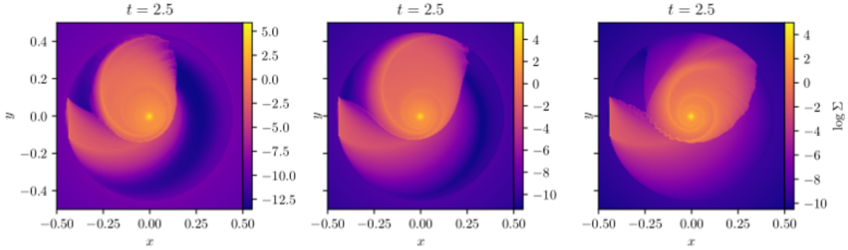
Simulations with canoP



Credit F. Drui (Phd, MDLS and ECP)

CanoP : self-gravitating accretion disk

Application to protoplanetary disk, N. Brucy, W. Verdier, M1 intership with P. Hennebelle, O. Iffrig, PK

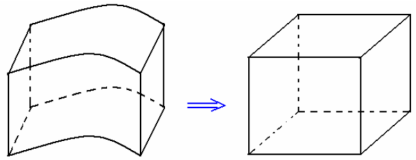


Conclusion

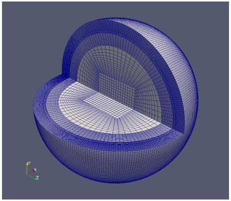
- **Gained expertise at designing / refactoring C++/CUDA applications using Kokkos**
  - much better global software design : **separation of concerns**
  - high-level concept (no CUDA), focus on **parallel computing pattern** (`for, reduce, scan, ...`)
  - data array access closely look like Fortran syntax
  - C++11 + template: a key to generic cleaner code
- **Developped new high-order num schemes: SDM**

Conclusion

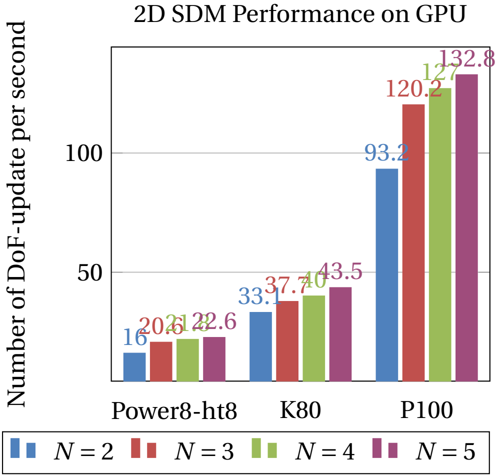
- **Futur developments: towards multi-architecture AMR with Kokkos**
- **adapt SDM scheme to spherical geometry via coordinate transformation + mesh refinement** (`CanoP`)
  $\partial_t Q + \partial_x F + \partial_y G + \partial_z H + M = 0 \Rightarrow \partial_t \tilde{Q} + \partial_\xi \tilde{F} + \partial_\eta \tilde{G} + \partial_\zeta \tilde{H} + \tilde{M} = 0$



- Implement SDM schemes for MHD
- **WIP: CanoP + Kokkos integration, make it available to all solvers**
- **towards global solar dynamo and surface physics with Sacha BRUN (CEA)**



2D SDM schemes - IBM Power8 vs Nvidia P100



- Time integration is RK3
- On average **Pascal P100** is $\times 2.8$ **to** $\times 3.0$ **faster** than **Kepler K80** (single GPU), no special optimization, just rebuild with architecture flags.
- **Pascal P100** is ~ $x5.8$ faster than **Power8 - HT8**
- by activating 8-way hyperthreading, Power8 version is 15 to 20% faster
- Performed on system `ouessant` at **IDRIS/GENCI**.