TASK-BASED PARALLELIZATION OF A FINITE VOLUME CODE FOR HYPERBOLIC CONSERVATION LAWS

ANR Solharis/ HPC-scalable-ecosystem day, 2022

Dr. Sangeeth Simon
- Team CAGIRE (V. Perrier, J. Jung, M. Haefele), INRIA Bordeaux Sud-Ouest & Laboratoire de Mathématiques et de leurs applications, Université de Pau et des Pays de l'Adour

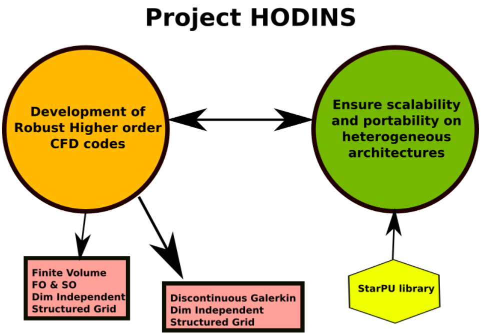
February 9, 2022

What is this research about?

- **The overall aim:** To have a highly scalable, hardware-agnostic, Higher Order, Discontinuous Galerkin - based, Navier-Stokes solver.
- **The problem:** How to *easily* develop scalable and portable applications for heterogeneous HPC clusters?
- **Our approach:** Use StarPU: a task-based parallelization paradigm to benefit from coexisting multi-core, multi-node & GPU architectures.
- **First milestone:** To implement a scalable, multidimensional, first & second order Finite Volume solver for the Euler flows using StarPU library.
- **Our findings so far:** Implementation of Finite Volume code (dimension independent (1D-2D-3D), first & second order accurate) complete. Good scalability behaviour observed in general.

Outline of the presentation

- Our motivation
- What we achieved on Project HODINS so far (Dim Independent code with first & second order accuracy).
- How we recovered scalability through code refactoring.
- How we performed Roofline modeling of the largest kernel.
- Conclusions.
- Our ongoing work (DG implementation + PPrime collaboration).


Project HODINS
Development of Robust Higher order CFD codes
Ensure scalability and portability on heterogeneous architectures
Finite Volume
FO & SO
Dim Independent
Structured Grid
Discontinuous Galerkin
Dim Independent
Structured Grid
StarPU library

Why Higher Order Methods for Nonlinear Balance laws?

Linear Acoustic equations

$$
\begin{aligned}
p_t + K_0 u_x &= 0 \\
\rho_0 u_t + p_x &= 0
\end{aligned}
\tag{1}
$$

- **Solutions that are superpositions** and hence comparitively easier to obtain.
- Computationally, they reduce to **global matrix operations** that can be optimized for an HPC application.

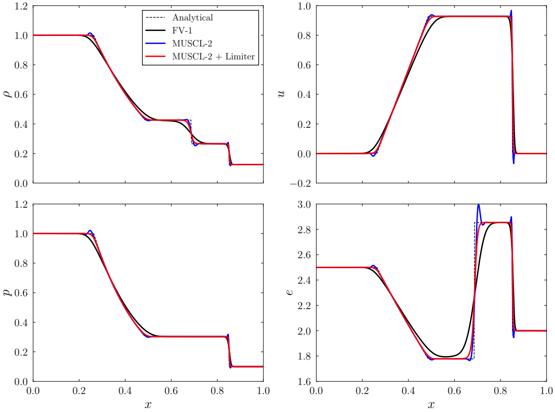
Why Higher Order Methods for Nonlinear Balance laws?

Euler equations

$$\begin{bmatrix} \rho \\ \rho u \\ E \end{bmatrix} + \begin{bmatrix} \rho u \\ \rho u^2 + p \\ (E+p)u \end{bmatrix} = 0 \qquad (1)$$

- **Develop discontinuities** as part of their solution.
- Obtaining interface fluxes at the element boundaries involves **nonlinear function evaluations**.
- Relatively **higher computational complexity** per time step.

Why Higher Order Methods for Nonlinear Balance laws?



- First order is 'diffusive' than second order.
- Need for limiting of solution near discontinuity for second order.

[1] Acknowledgement: Carbajal-Carrasco Luis, Institut P'(CNRS)

Why Higher Order Methods for Nonlinear Balance laws?

Objective of any higher order method

Obtaining better resolution on coarser grids compared to a lower order method!6

Higher order CFD methods

Finite Difference, **Finite Volume** and **Finite Element**.

HODINS-v1

[1]HODINS-v1-2D → a C++-based, two-dimensional, first order accurate, Finite Volume code for the Euler equations using StarPU.

[1]Essadki, M. Jung, J. Larat, A. Pelletier, M. Perrier, V. A Task-Driven Implementation of a Simple Numerical Solver for Hyperbolic Conservation Laws. *ESAIM: ProcS* (2018) **63**:228–247.

HODINS-v1

- Domain decomposition: One task per partition

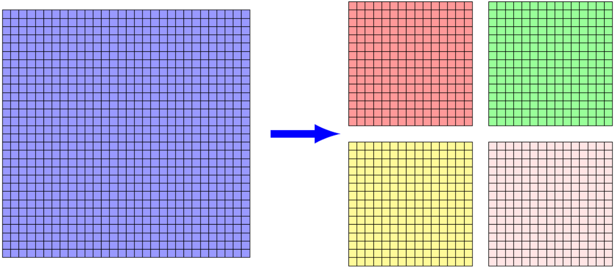


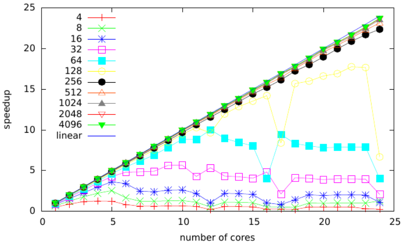
FIGURE 1. Partitioning of an initial mesh of $\{\texttt{Nx} = 30 \times \texttt{Ny} = 30\}$ cells into $\{\texttt{NPartX} = 2 \times \texttt{NPartY} = 2\}$ 225 cells domains.

*Taken from [1]

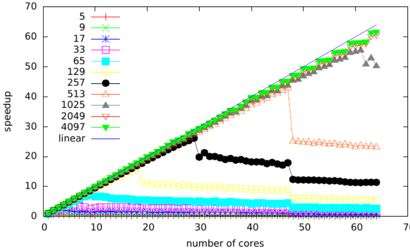
[1] Essadki, M. Jung, J. Larat, A. Pelletier, M. Perrier, V. A Task-Driven Implementation of a Simple Numerical Solver for Hyperbolic Conservation Laws. *ESAIM: ProcS* (2018) **63**:228–247.

HODINS-v1-2D:Major learnings

◎ **Importance of tasksizeOverhead**[1] Eager scheduler used. Legends are in (μs).



(A) 2 dodeca-core Haswell Intel Xeon E5-2680.



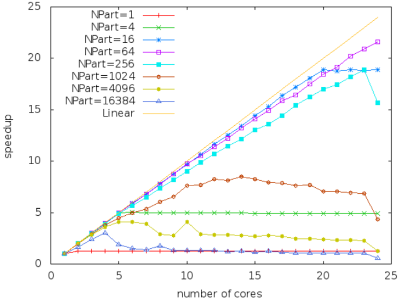
(B) Xeon Phi KNL.

(a) Haswell (b) KNL

Tasks must have a minimum execution time (~1ms) for scalability from StarPU!

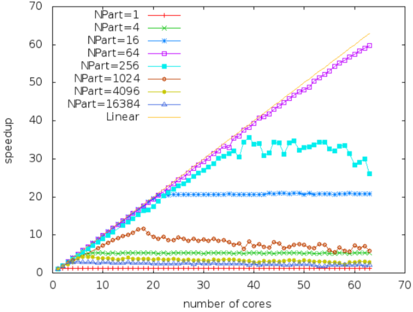
HODINS-v1-2D:Major learnings

- **Importance of numerous large tasks**[1] Results for 1024×1024 grid.



(A) 2 dodeca-core Haswell Intel Xeon E5-2680.

(a) Haswell



(B) Xeon Phi KNL.

(b) KNL

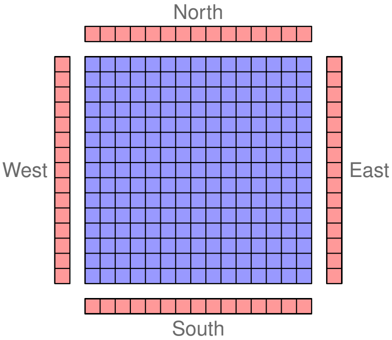
For scalability from StarPU: Numerous tasks ($\mathcal{O}$(No of Cores)) that are individually large enough (>tasksizeoverhead)

Taking HODINS forward

Code refactoring to improve HODINS

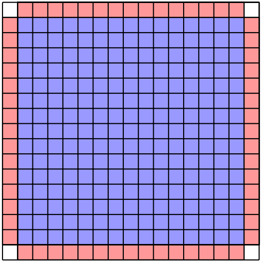
- ✓ Making the code dimension independent through templating → enable 1D, 2D, 3D cases.
- ✓ Restructuring memory management → code simplification and reusability.
- ✓ Extension to 3D → improving arithmetic intensity.
- ✓ Extension to Second order using SSP-RK MUSCL → improving arithmetic intensity.
- ✓ Major kernel rewriting to promote faster computations.
- ✓ Promote code reusability → Reuse internal residual kernel loops for border residual computation.

Restructuring data handles



- 5 pointers
- 5 starpu_data_handle
- 1st step: launch concurrently
  - Internal residual computation[R]
  - copy inside overlaps[W]
- 2nd step: compute the border residual, involving both internal[R] and borders[R] data handles
  - But matching indices between internal and overlaps is not straightforward to implement

Restructuring data handles



- 1 pointers
- 6 `starpu_data_handle`
  - One view with Internal and overlaps
  - One view with full pointer
- 1st step: launch concurrently
  - Internal residual computation
  - copy inside overlaps
- 2nd step: compute the border residual, involving the full view of the pointer
  - Use the same algorithm as for internal residual, but with different indices.

Extending to second order accuracy

MUSCL Scheme

$$\frac{d\mathbf{u}_i}{dt} + \frac{1}{\Delta\Omega}\mathbf{F}(\mathbf{u}^*_{i+1/2}) - \mathbf{F}(\mathbf{u}^*_{i-1/2}) = 0$$

$$\mathbf{u}^*_{i\pm1/2} = \mathbf{u}^*_{i\pm1/2}(\mathbf{u}^L_{i\pm1/2}, \mathbf{u}^R_{i\pm1/2})$$

$$\mathbf{u}^L_{i+1/2} = \mathbf{u}_i + \frac{1}{2}\phi(r_i)(\mathbf{u}_{i+1} - \mathbf{u}_i)$$

$$\mathbf{u}^R_{i+1/2} = \mathbf{u}_{i+1} - \frac{1}{2}\phi(r_{i+1})(\mathbf{u}_{i+2} - \mathbf{u}_{i+1})$$

$$r_i = \frac{\mathbf{u}_i - \mathbf{u}_{i-1}}{\mathbf{u}_{i+1} - \mathbf{u}_i}$$

where the function $\phi(r)$ works as a slope limiter and ensures that the solution is TVD. We use a **min-mod** limiter function

Extending to second order accuracy

Strong-Stability Preserving Runge Kutta Scheme

$$\begin{aligned} \mathbf{u}^1 &= \mathbf{u}^n + \Delta t \mathbf{R}(\mathbf{u}^n) \\ \mathbf{u}^{n+1} &= \frac{1}{2}\mathbf{u}^n + \frac{1}{2}\mathbf{u}^1 + \frac{1}{2}\Delta t \mathbf{R}(\mathbf{u}^1) \end{aligned} \tag{1}$$

Major Kernel modifications: Making the CELL class lighter

Instead of...

```
class Cell
{
  double w[5];
  double primitive[5];
}
```

We have...

```
template<int nDIM>
class CELL
{
    public:
        double w[nDIM+2];
}
```

Major Kernel modifications: Getting rid of large mallocs

Instead of...

```
int ArraySize = ldu*Ny;
double* drhobydx = (double*)malloc(ArraySize*sizeof(double));
double* dubydx   = (double*)malloc(ArraySize*sizeof(double));
double* dvbydx   = (double*)malloc(ArraySize*sizeof(double));
double* dPbydx   = (double*)malloc(ArraySize*sizeof(double));
double* drhobydy = (double*)malloc(ArraySize*sizeof(double));
double* dubydy   = (double*)malloc(ArraySize*sizeof(double));
double* dvbydy   = (double*)malloc(ArraySize*sizeof(double));
double* dPbydy   = (double*)malloc(ArraySize*sizeof(double));
```

We have...

```
void getPrimitiveFromConservative(): input(conserved var), output(primitive var)
double computeDerivatives(): input(primitive var on stencil),
output(limited derivative)
```

Major Kernel modifications: Replacing divisions by Multiplications

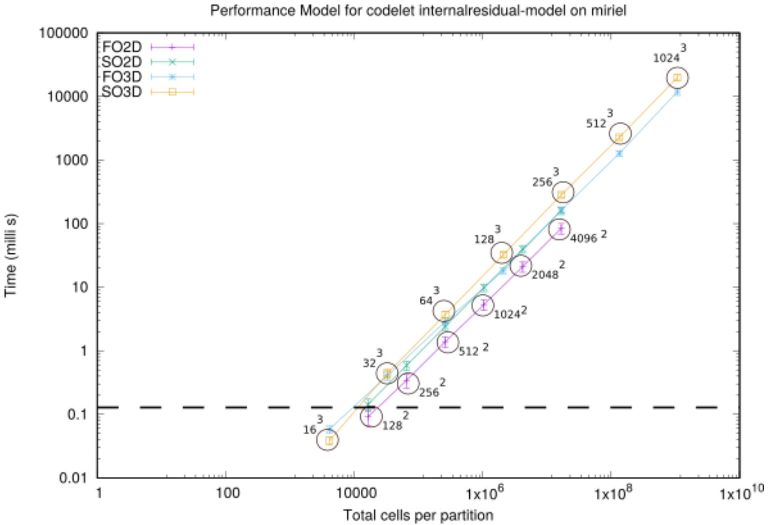
Instead of...

```
Fl[0] = n[0]*Ul.w[1] + n[1]*Ul.w[2] + n[2]*Ul.w[3];
Fr[0] = n[0]*Ur.w[1] + n[1]*Ur.w[2] + n[2]*Ur.w[3];
Fl[1] = Ul.get_u()*Fl[0] + Ul.get_p()*n[0];
Fr[1] = Ur.get_u()*Fr[0] + Ur.get_p()*n[0];
Fl[2] = Ul.get_v()*Fl[0] + Ul.get_p()*n[1];
Fr[2] = Ur.get_v()*Fr[0] + Ur.get_p()*n[1];
Fl[3] = Ul.get_w()*Fl[0] + Ul.get_p()*n[2];
Fr[3] = Ur.get_w()*Fr[0] + Ur.get_p()*n[2];
Fl[4] = (Ul.get_e() + Ul.get_p()/Ul.w[0])*Fl[0];
Fr[4] = (Ur.get_e() + Ur.get_p()/Ur.w[0])*Fr[0];
alpha = fmax(Ul.getMaxSpeedWave(n), Ur.getMaxSpeedWave(n));
```

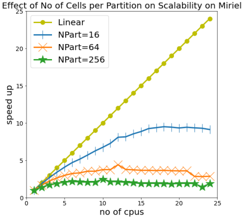
We have...

```
double rhoInverseL = 1.0/Ul.w[0];
double rhoInverseR = 1.0/Ur.w[0];
Fl[i] = (Ul.w[i]*rhoInverseL)*Fl[0] + Ul.get_p(rhoInverseL)*n[i-1];
Fr[i] = (Ur.w[i]*rhoInverseR)*Fr[0] + Ur.get_p(rhoInverseR)*n[i-1];
Fl[nDIM+1] = (Ul.get_e(rhoInverseL) + Ul.get_p(rhoInverseL)*rhoInverseL)*Fl[0];
Fr[nDIM+1] = (Ur.get_e(rhoInverseR) + Ur.get_p(rhoInverseR)*rhoInverseR)*Fr[0];
alpha = fmax(Ul.getMaxSpeedWave(n,rhoInverseL), Ur.getMaxSpeedWave(n, rhoInverse
```

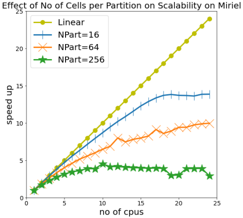
Codelet Internal Residual comparison between 2D, 3D, FO and SO



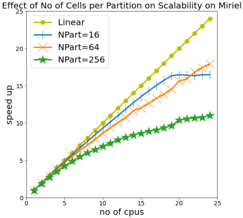
Strong scalability: First Order, 2D, Miriel



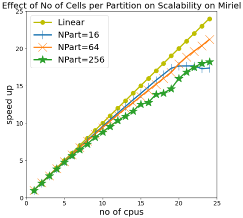
(a) GRID=256^2



(b) GRID=512^2

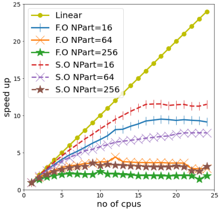


(c) GRID=1024^2

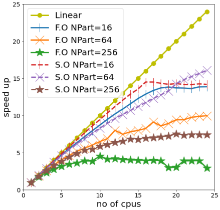


(d) GRID=2048^2

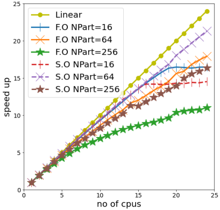
Strong scalability: First & Second Order, 2D, Miriel



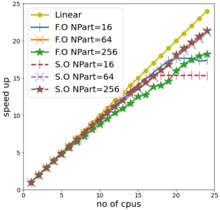
(a) GRID=256^2



(b) GRID=512^2

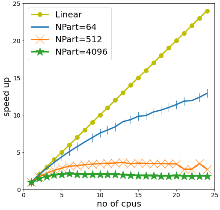


(c) GRID=1024^2

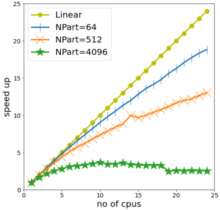


(d) GRID=2048^2

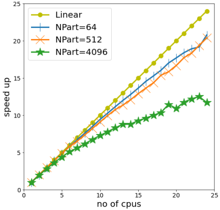
Strong scalability: First Order, 3D, Miriel



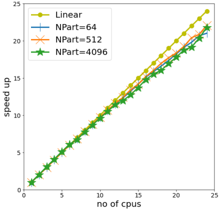
(a) GRID=64^3



(b) GRID=128^3

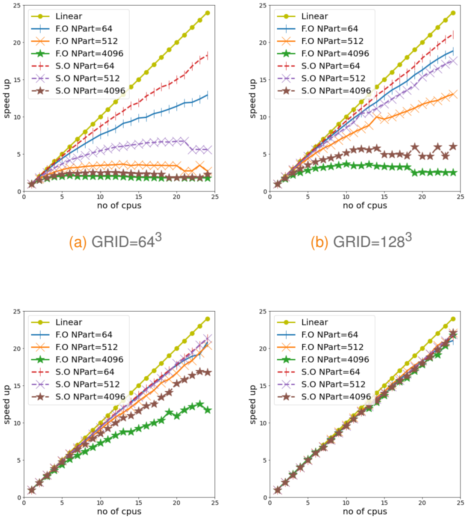


(c) GRID=256^3



(d) GRID=512^3

Strong scalability: First & Second Order, 3D, Miriel



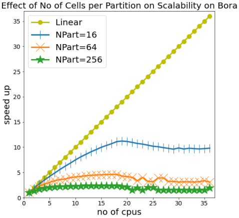
(a) GRID=64^3

(b) GRID=128^3

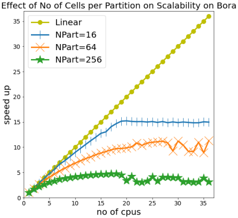
(c) GRID=256^3

(d) GRID=512^3

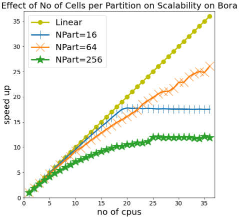
Strong scalability: First Order, 2D, Bora



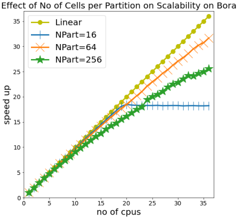
(a) GRID=256^2



(b) GRID=512^2

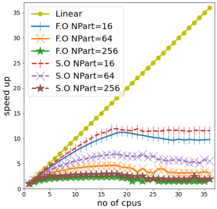


(c) GRID=1024^2

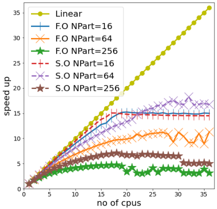


(d) GRID=2048^2

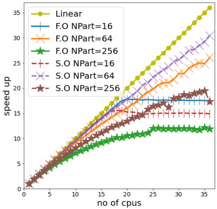
Strong scalability: First & Second Order, 2D, Bora



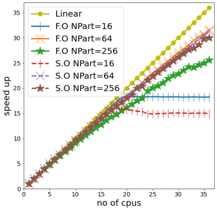
(a) GRID=256^2



(b) GRID=512^2

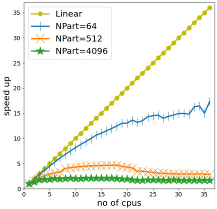


(c) GRID=1024^2

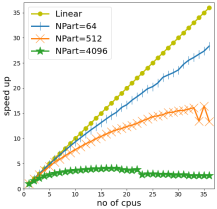


(d) GRID=2048^2

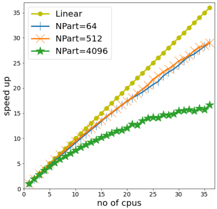
Strong scalability: First Order, 3D, Bora



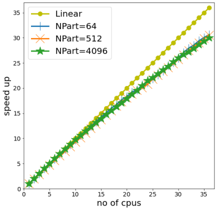
(a) GRID=64^3



(b) GRID=128^3

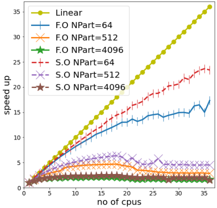


(c) GRID=256^3

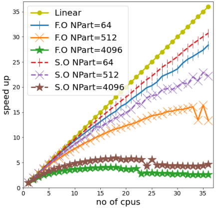


(d) GRID=512^3

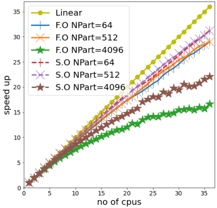
Strong scalability: First & Second Order, 3D, Bora



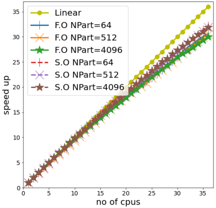
(a) GRID=64^3



(b) GRID=128^3



(c) GRID=256^3



(d) GRID=512^3

Roofline model for Internal Residual Kernel

- Idea → Roofline plot of internal residual kernel on Miriel node.
- Miriel Characteristics → Max bandwidth[2] = 68 (GB/s) & Max Flop Count[3] = 480 GFLOPS/s.

Arithmetic Intensity (abscissa)

Requires cache models to estimate Kernel FLOPS & Data Consumption.

GFlops/s (ordinate)

Measured from StarPU Codelet Performance model for the Kernel.

[2] STREAM benchmark

[3] Number of cores * Avg freq * Number of AVX oper * Number of FMA oper

Cache models [4]: No Cache & Infinite Cache

Assumptions

- Machine model: Processor & two memory levels: limited fast (last-level shared cache) & unlimited slow (DRAM).
- Two way data transfer with overwrite.
- All computations only on the data in the fast level.
- All analysis for an isolated partition comprising of NX $\times$ NY $\times$ NZ cells. Ghost cell impacts not considered.
- Finite Volume algorithm decomposed into its essentials: solution reconstruction, Riemann solver, Residual computation etc.
- Total Flops = Flops per interface * Number of interfaces. Remains same for all cache models.
- Only addition and multiplication are counted. Special operations sqrt(), pow() etc excluded. No vectorization.
- Total Bytes = Bytes per interface * Number of interfaces. Changes with cache models.

[4] J. Loffeld and JAF. Hittinger, On the arithmetic intensity of high-order finite-volume discretizations for hyperbolic systems of conservation laws, The International Journal of High Performance Computing Applications, 2019, Vol. 33(1) 25–52

Cache models [4]: No Cache & Infinite Cache

No Cache

- Degenerate case → Data loaded for computations concerning each interface.
- Provides upper bound on memory operations for kernel.
- Provides lower bound on arithmetic intensity for kernel.
- Major cost for data handles. Both reading & writing are included.
- Assume registers capable of storing local variables.

Infinite Cache

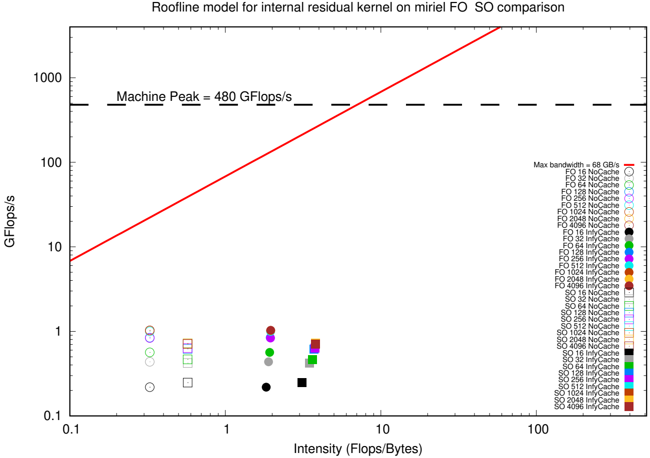
- Ideal case → Data once loaded for an interface stays in the fast memory.
- Provides lower bound on memory operations for kernel.
- Provides upper bound on arithmetic intensity for kernel.
- Major cost in reading and writing the entire grid worth of data once.

[4] J. Loffeld and JAF. Hittinger, On the arithmetic intensity of high-order finite-volume discretizations for hyperbolic systems of conservation laws, The International Journal of High Performance Computing Applications, 2019, Vol. 33(1) 25–52

Results: FO & SO comparison for No Cache and Infy Cache

2D Results

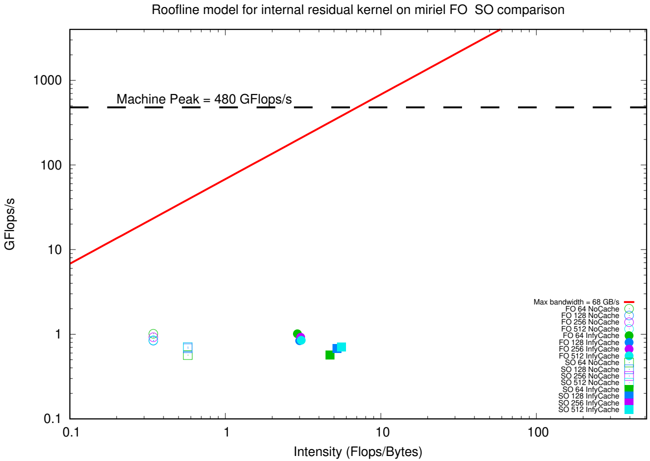
(Total NPART=16 & NCPU=16)



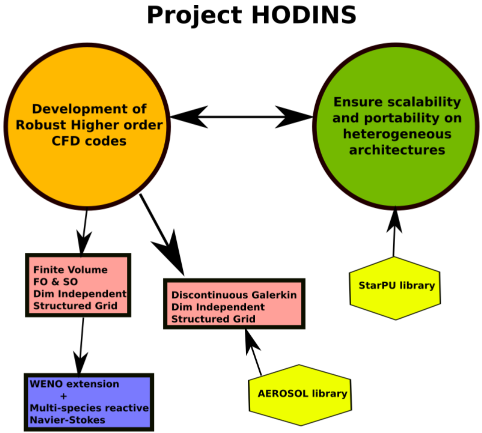
Results: FO & SO comparison for No Cache and Infy Cache

3D Results

(Total NPART=64 & NCPU=24)



Why Discontinous Galerkin method?



Why Discontinous Galerkin method?

- **Best of both frameworks** → Finite Element (polynomial representation of solution on elements & matrix operations) & Finite Volume (conservation, interface fluxes, limiters).
- **Good data locality & compact stencil** → Each element is equipped with enough data (except interface fluxes) to compute its own residuals. Each element interacts only with its immediate neighbours.
- **Local matrices instead of global matrices** → In DG we completely avoid global matrix operations.
- **Diffusion operator adds more computations** → Evaluation of second order gradients using same DOF.
- **Enhanced arithmetic intensity** → Each element ideally does more computation with local data compared to communication.

Conclusion

- We motivated the **need for H.O. methods for Non-Linear equations**
- We learned from HODINS-v1-2D-First order that **saturating task heap with compute intense tasks is necessary for achieving scalability**.
- We **built HODINS-v2 which is a dimension-independent, second order accurate code with reconfigured memory management and faster kernels**.
- We **studied strong scalability for HODINS-v2-2D,3D-First order, Second order.**Improving order of accuracy and dimension has positive impact on scalability.
- We **performed roofline modeling of the internal residual kernel**. Improving order of accuracy and dimension has positive impact on improving the Arithmetic Intensity.
- We **discussed some interesting extension of HODINS** currently underway.

My sincere thanks to:

Inria & "Conseil Régional Nouvelle Aquitaine".
Team CAGIRE (Vincent Perrier, Jonathan Jung & Matthieu Hafele)
Samuel Thibault & Mathieu Faverge for their useful discussions on StarPU and Vite.
Luis Carrasco Carbajal for the shock tube results.
ANR Solharis / Région Nouvelle Aquitaine hpc-scalable-ecosystem Organizers.