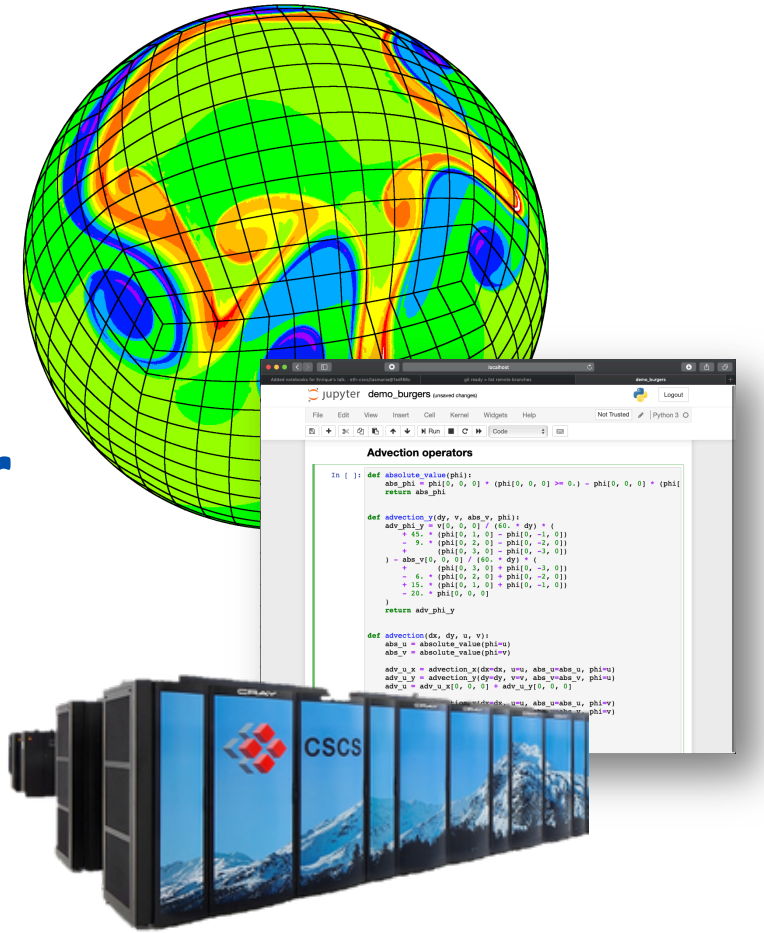
High Performance Computing for Weather and Climate (HPC4WC)

Content: Introduction
Lecturer: Oliver Fuhrer
Block course 701-1270-00L
Summer 2020

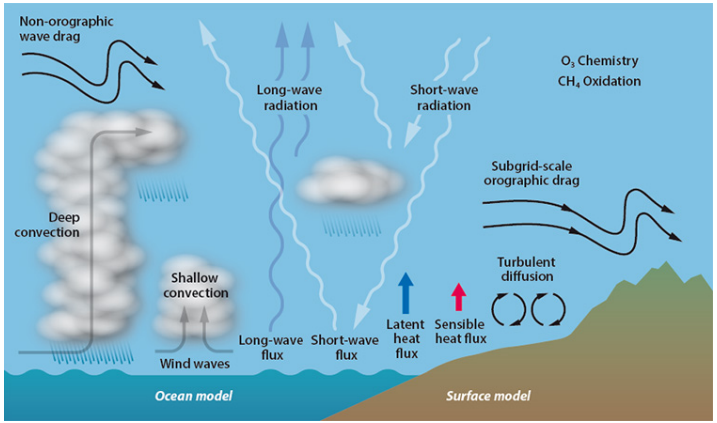


Learning goals

- Understand why weather and climate simulation requires high-performance computing (HPC)
- Understand why stencil computations are a key algorithmic motif of weather and climate models
- Understand a simple stencil program: higher-order monotonic diffusion (Xue 2000, MWR)

Modeling the Earth system

Physical understanding



Governing equations (e.g. atmosphere)

$$\frac{d}{dt}\mathbf{v} = -2\boldsymbol{\Omega} \times \mathbf{v} - \frac{1}{\rho}\nabla_3 p + \mathbf{g} + \mathbf{F}$$ Conservation of momentum (Navier-Stokes)

$$C_v \frac{d}{dt}(\rho q) + p\frac{d}{dt}\left(\frac{1}{\rho}\right) = J$$ Conservation of energy (1st Law of Thermodynamics)

$$\frac{\partial}{\partial t}(\rho) = -\nabla_3 \cdot (\rho \mathbf{v})$$ Conservation of air mass

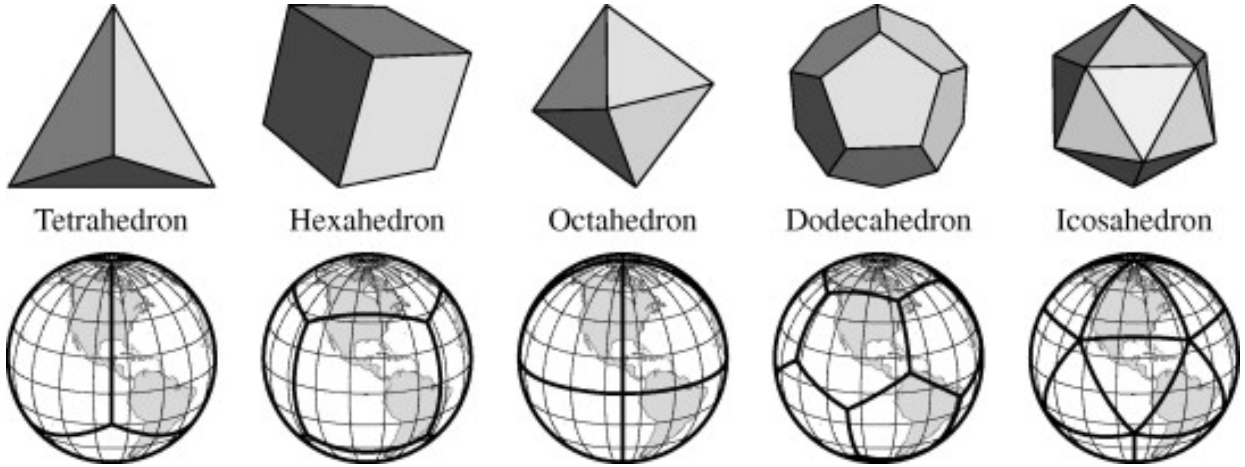
$$\frac{\partial}{\partial t} = -\nabla_3 \cdot (\rho \mathbf{v} q) + \rho (E - C)$$ Continuity of water vapor mass

$$p = \rho R T$$ Equation of state (Ideal gas law)

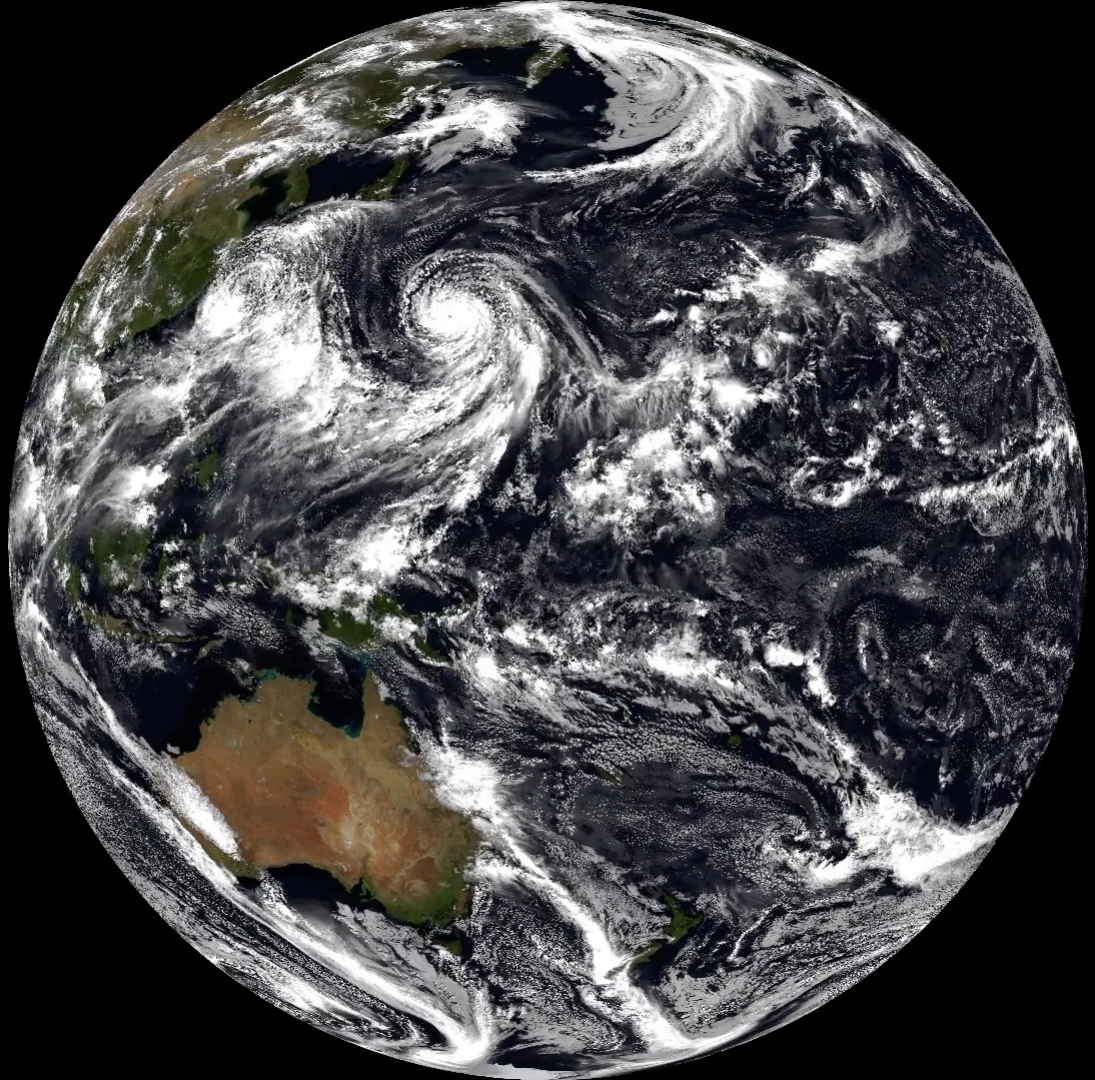
Variables: $\{\mathbf{v}, p, T, \rho, q\}$

(Image source https://www.ecmwf.int/sites/default/files/medialibrary/2017-09/atmospheric-physics-754px.jpg)

Grids on a sphere



(Image source https://www.cosmic-core.org/free/article-42-geometry-platonic-solids-part-3-spherical-stereographic-solids/

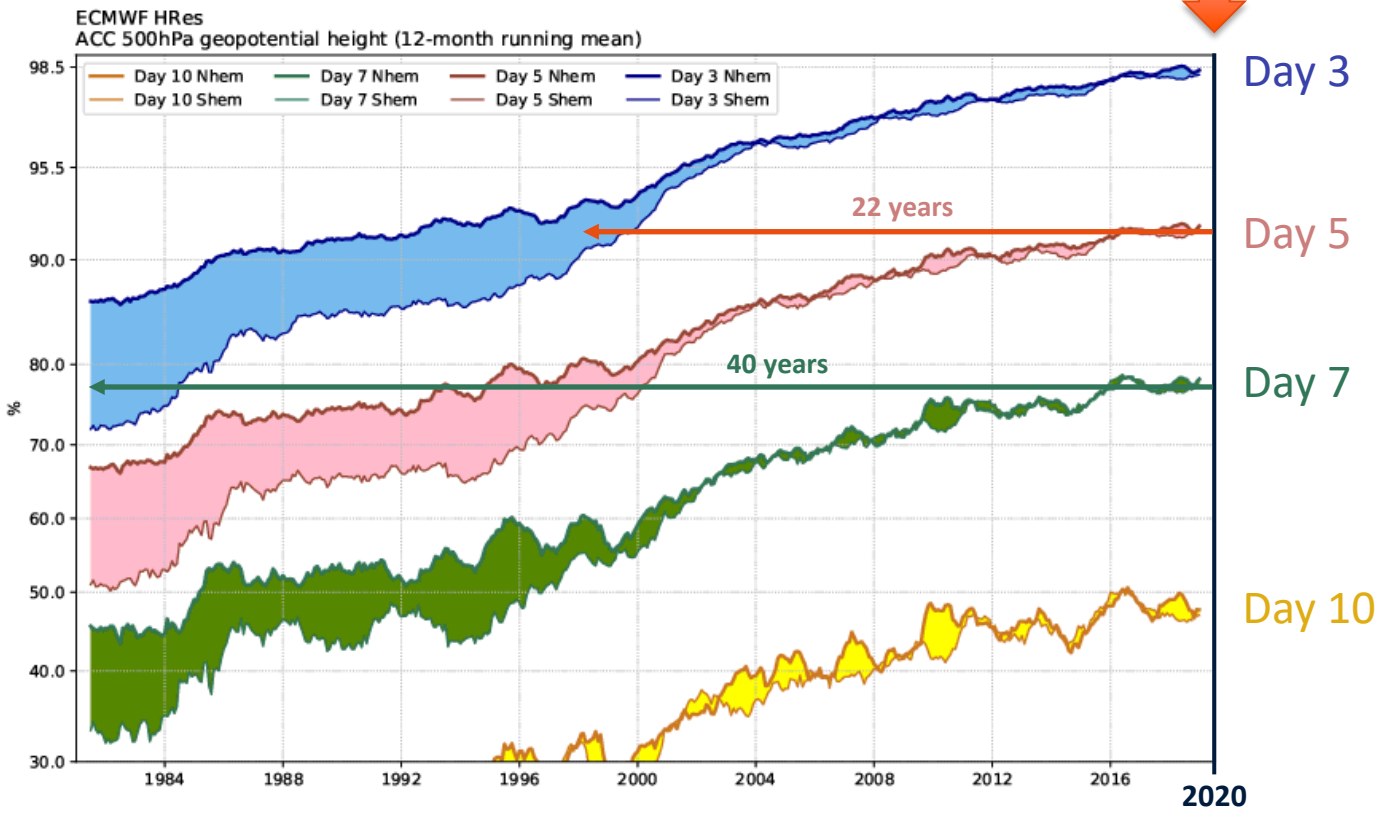


2016-08-11 18:00Z
258 Forecast Hours
FV3 3km

Visualization
Xi Chen@FV3 team

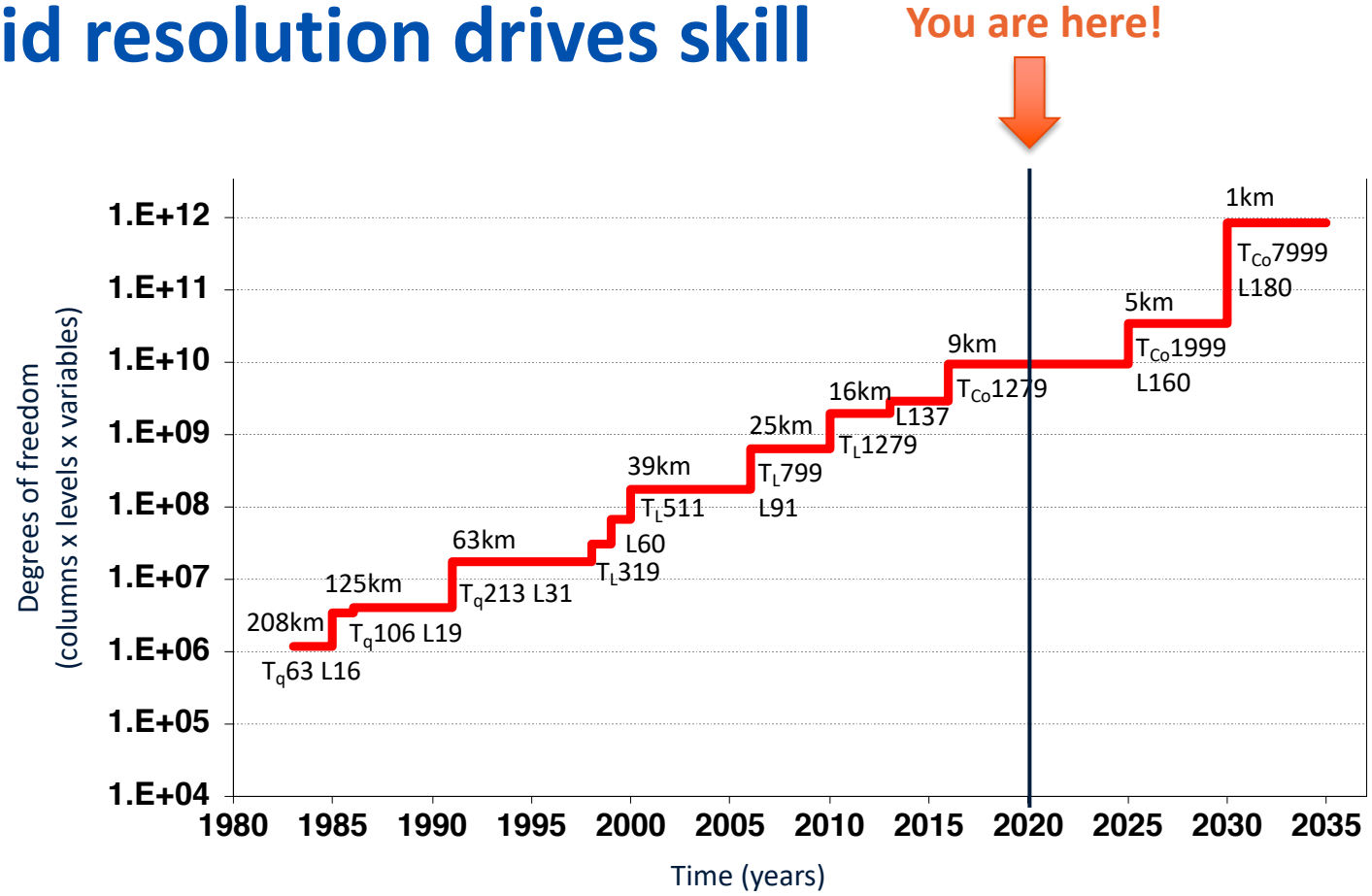
Forecast skill over time

You are here!



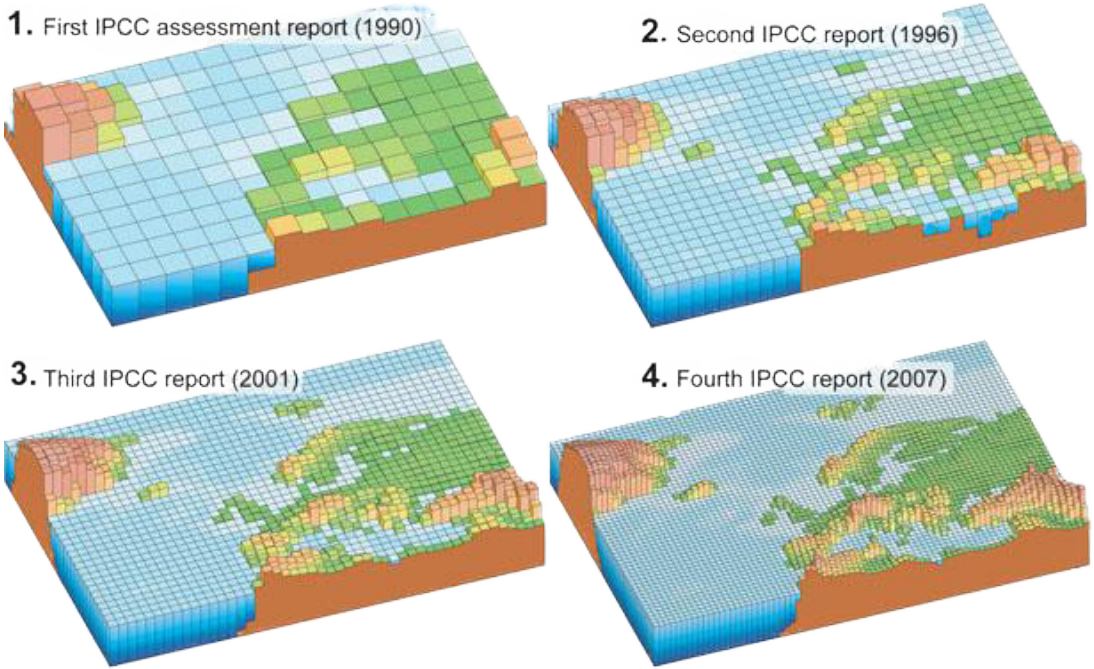
(Image source https://www.ecmwf.int/en/forecasts/charts/catalogue/plwww_m_hr_ccaf_adrian_ts?facets=undefined&time=2020051100)

Grid resolution drives skill

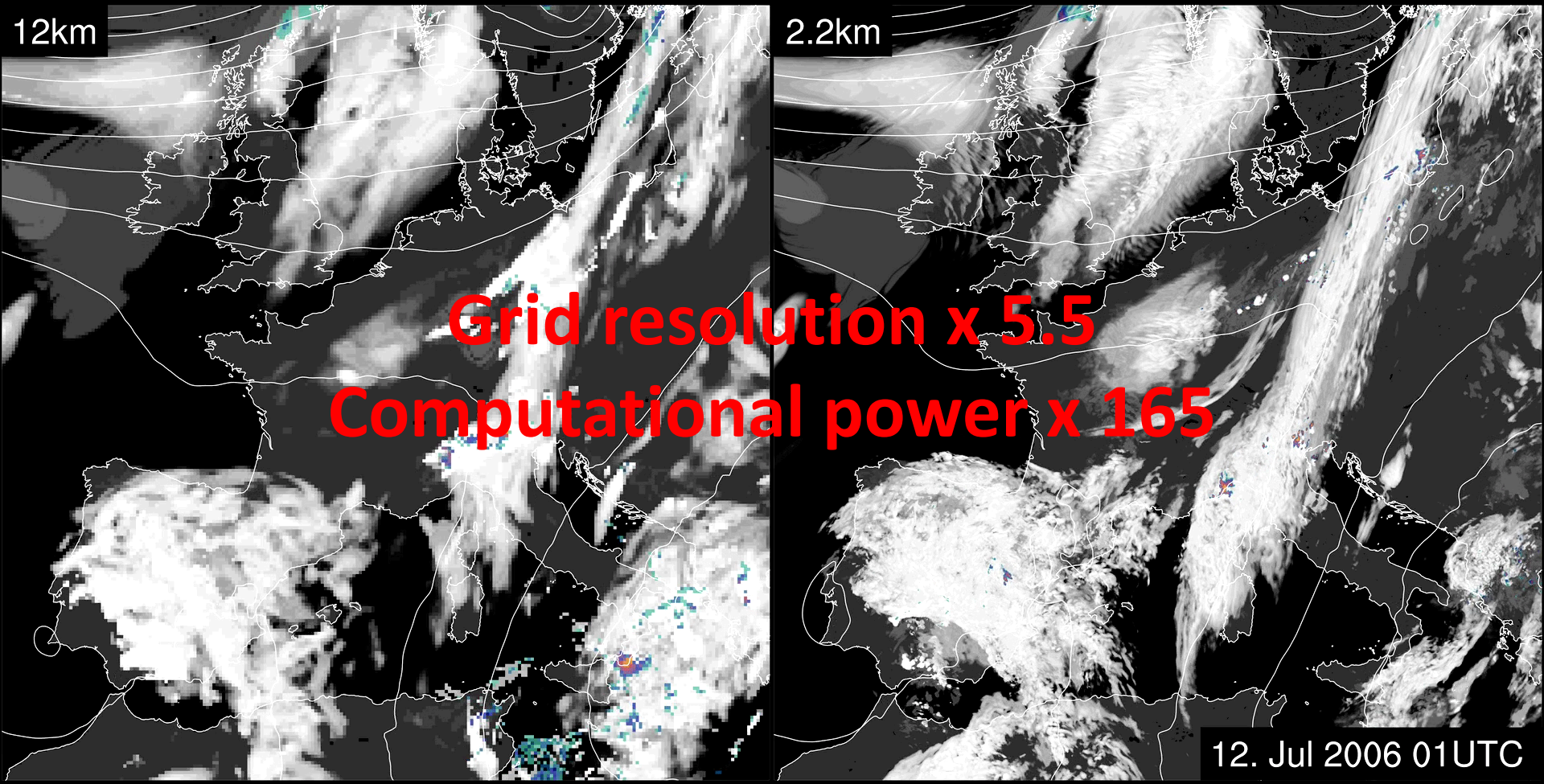


(Image source Schulthess et al. 2019, IEEE Race to Exascale Computing)

Grid resolution in the IPCC



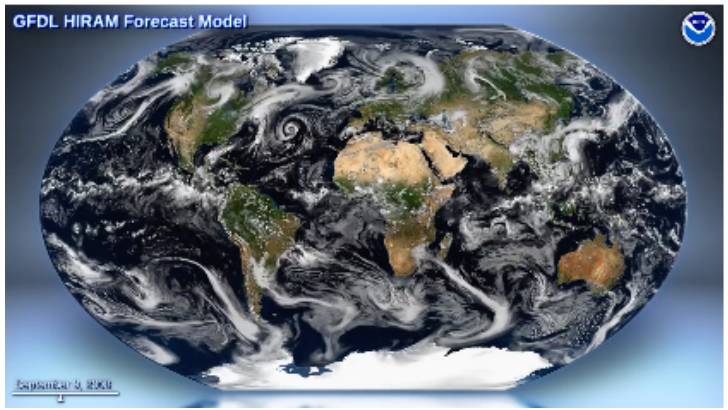
(Image source https://nas-sites.org/climate-change/climatemodeling/common/images/image_page_3_2_resolution.png


12km
2.2km
Grid resolution x 5.5
Computational power x 165
12. Jul 2006 01UTC

Example applications

- Swiss national weather forecast (COSMO-1)
  1152 x 774 x 80 gridpoints
  Δx = 1.1 km
  Δt = 10 s
  80x faster than real time

- Global climate model with 3 km resolution (DYAMOND)
  57 x 10^6 x 80 gridpoints
  Δx = 3.0 km
  Δt = 9 s
  1000x faster than real time



Why HPC?

Intel Xeon E5-2690 v3
@2.6 Ghz, 135 W

- Swiss national weather forecast (COSMO-1)
  1152 x 774 x 80 gridpoints
  Δx = 1.1 km
  Δt = 10 s
  80x faster than real time

~ 750 CPUs

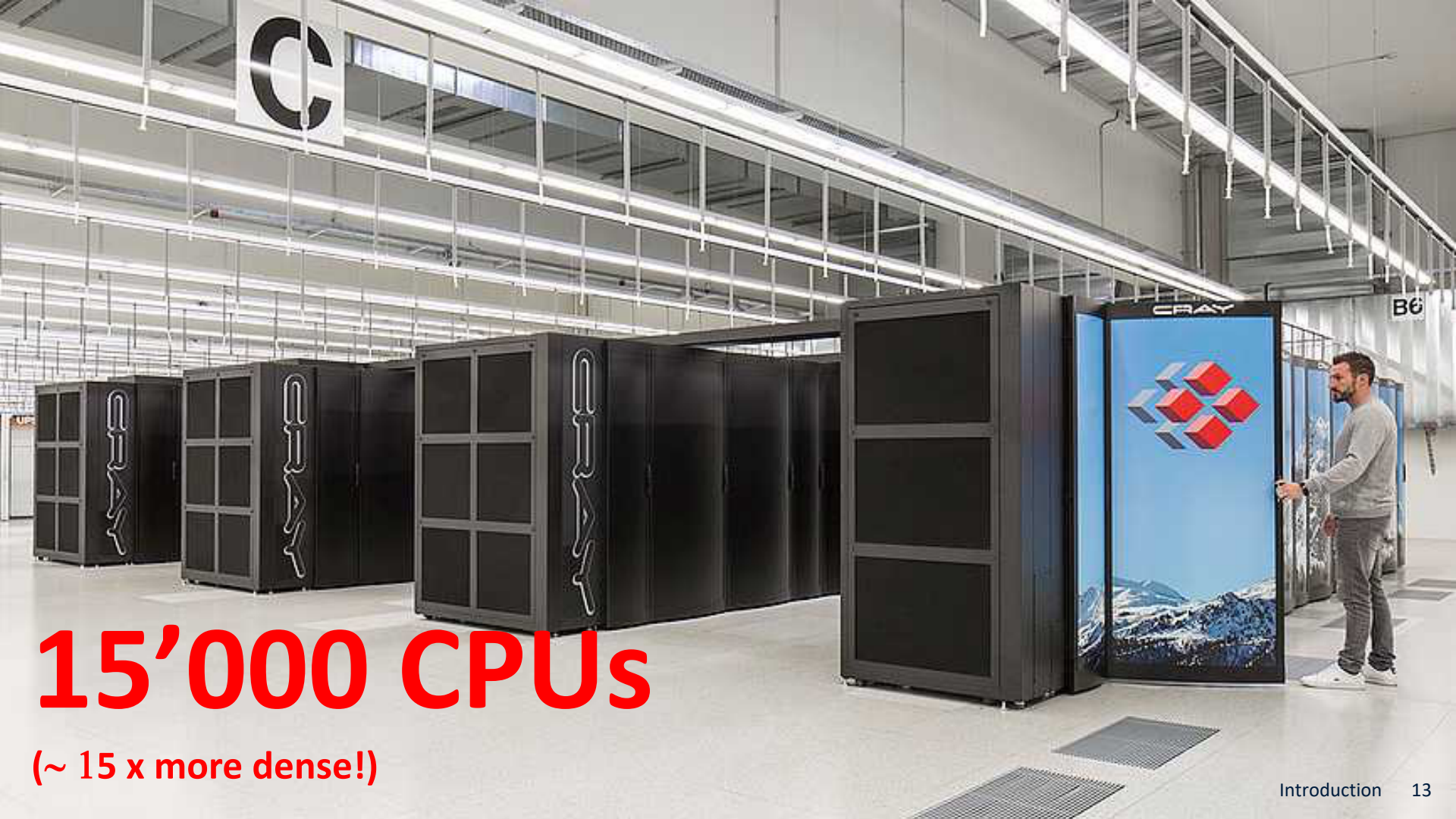
~ 100 kW (= 130 households)

- Global climate model with 3 km resolution (DYAMOND)
  57 x 10^6 x 80 gridpoints
  Δx = 3.0 km
  Δt = 9 s
  ~~1000x faster than real time~~
  Currently only 100x feasible!

~ 750'000 CPUs

~ 100 MW (= 10% of Gösgen)


KLAT2
KAOS
Compilers
Hardware
Architectures
Operating
Systems
~130 CPUs

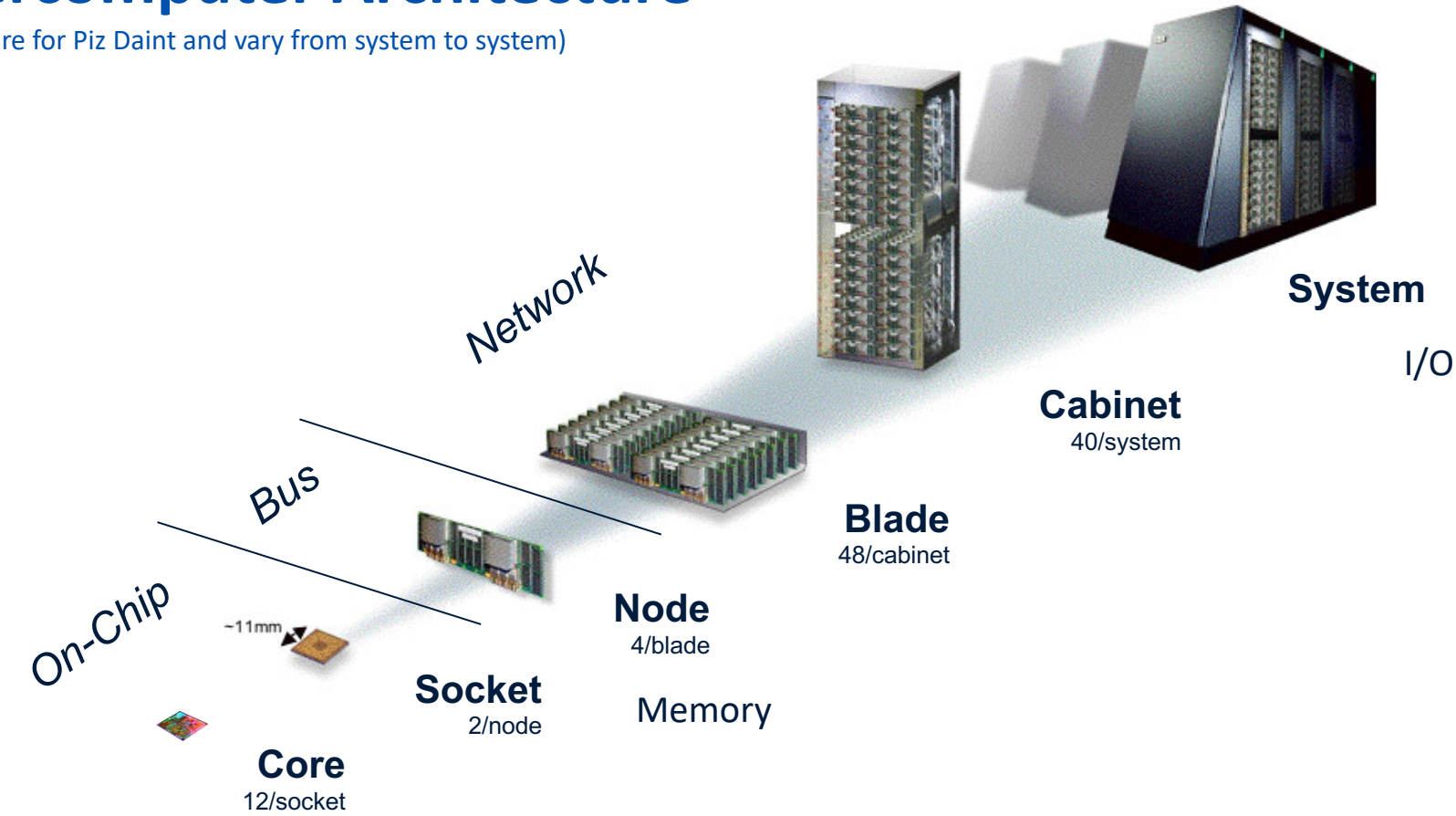


15’000 CPUs

(~ 15 x more dense!)

Supercomputer Architecture

(Numbers are for Piz Daint and vary from system to system)



> *High Performance Computing* is the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation in order to solve large problems in science, engineering, or business.

Linear 2D Advection

- Governing equation
  (on a square domain $L\,x\,L$)

$$s_t + u s_x + v s_y = 0$$
$$s = s(x, y, t) \quad u = u(x, y) \quad v = v(x, y)$$
$$x, y \in [0, L] \quad t > 0$$

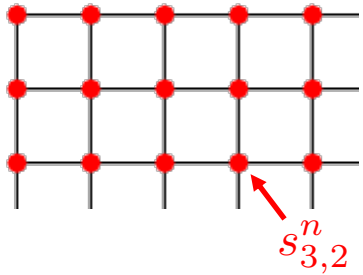
- Initial and boundary condition

$$s(x, y, t) = s_0(x, y, 0)$$
$$s(x + nL, y + mL, t) = s(x, y, t) \quad \forall \quad n, m \in \mathbb{Z}$$

- Many discretization methods to choose from…
  - Finite differences (FDM)
  - Finite volumes (FVM)
  - Finite element (FEM)
  - Spectral
  - …

Finite difference method (FDM)

- Store values at gridpoints



$$x_{i,j} = i\,\Delta x \quad y_{i,j} = j\,\Delta y$$
$$t^n = n\,\Delta t$$
$$s^n_{i,j} = s(x_{i,j}, y_{i,j}, t^n)$$

- Spatial discretization
  (3rd-order upstream)

$$\partial_x s_{i,j} = \frac{1}{6\Delta x} \begin{cases} 2s_{i+1,j} + 3s_{i,j} - 6s_{i-1,j} + s_{i-2,j} & u_{i,j} > 0 \\ -s_{i+2,j} + 6s_{i+1,j} - 3s_{i,j} - 2s_{i-1,j} & u_{i,j} > 0 \end{cases}$$

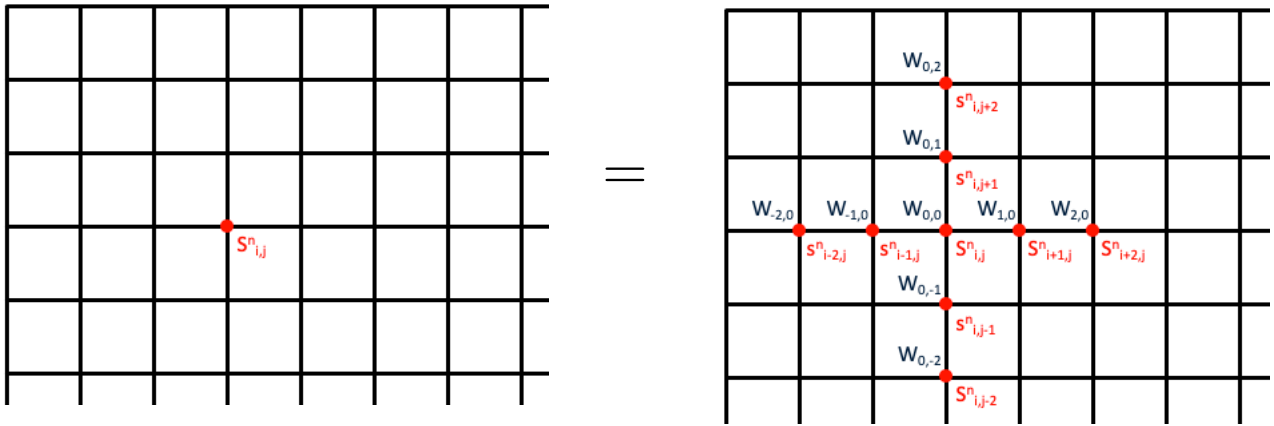
$$\partial_y s_{i,j} = \dots$$

- Time discretization
  (Euler)

$$\partial_t s^n = \frac{s^{n+1} - s^n}{\Delta t}$$

Stencil computation (5x5, sparse)

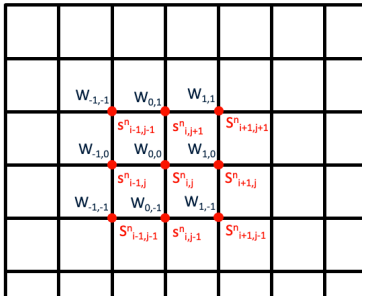
$$s_{i,j}^{n+1} = \sum_{i_{\mathrm{rel}},j_{\mathrm{rel}}} w_{i_{\mathrm{rel}},j_{\mathrm{rel}}}\, s_{i+i_{\mathrm{rel}},j+j_{\mathrm{rel}}}^{n}$$



weighted sum of grid values
compact neighborhood
same pattern ∀ gridpoints

“A *stencil computation* is an algorithmic motif where the value at a certain grid point is computed from a compact neighborhood of gridpoints in it’s vicinity on the computational grid using the same pattern for every gridpoint.”

Stencil program (3x3)




```
# weights stored in 3x3 array
for j = 1, nj
  for i = 1, ni
    s_new(i, j) = &
      w(-1, -1) * s(i-1, j-1) + w(0, -1) * s(i, j-1) + w(1, -1) * s(i+1, j-1)  &
      w(-1,  0) * s(i-1, j  ) + w(0,  0) * s(i, j  ) + w(1,  0) * s(i+1, j  )  &
      w(-1,  0) * s(i-1, j+1) + w(0,  1) * s(i, j+1) + w(1,  1) * s(i+1, j+1)
```

```
# weights literal constants
for j = 1, nj
  for i = 1, ni
    s_new(i, j) = &
      0.125 * s(i-1, j-1) + 0.25 * s(i, j-1) + 0.125 * s(i+1, j-1)  &
      0.25  * s(i-1, j  ) + 1.00 * s(i, j  ) + 0.25  * s(i+1, j  )  &
      0.125 * s(i-1, j+1) + 0.25 * s(i, j+1) + 0.125 * s(i+1, j+1)
```

Not stencils

- (Some) implicit methods (e.g. vertical advection)
- Spectral methods (e.g. IFS dynamical core)
- Other algorithmic motifs
  - Reductions
  - Searches
  - …

Summary

- Spatial resolution drives required computational power $1/\Delta x^3$, this is why we need supercomputers for modeling weather and climate.
- While models use a myriad of different numerical techniques to solve the governing PDEs, the main algorithmic motif resulting from these methods are stencil computations.

Our course will **focus exclusively on stencil computations**, but the concepts are more general and transferable.

Higher-order diffusion (Xue 2000, MWR)

- Atmospheric and ocean models often need some form of numerical filtering to control small-scale noise.
- Xue 2000 (Monthly Weather Review) published a class of higher-order monotonic filters that are frequently used.
- We will focus on the 4th-order non-monotonic diffusion for this course.

AUGUST 2000 XUE 2853

High-Order Monotonic Numerical Diffusion and Smoothing

MING XUE

Center for Analysis and Prediction of Storms, University of Oklahoma, Norman, Oklahoma

(Manuscript received 28 May 1999, in final form 17 September 1999)

ABSTRACT

High-order numerical diffusion is commonly used in numerical models to provide scale selective control over small-scale noise. Conventional high-order schemes have undesirable side effects, however: they can introduce noise themselves. Two types of monotonic high-order diffusion schemes are proposed. One is based on flux correction/limiting on the corrective fluxes, which is the difference between a high-order (fourth order and above) diffusion scheme and a lower-order (typically second order) one. Overshooting and undershooting found in the solutions of higher-order diffusions near sharp gradients are prevented, while the highly selective property of damping is retained.

The second simpler (flux limited) scheme simply ensures that the diffusive fluxes are always downgradient; otherwise, the fluxes are set to zero. This much simpler scheme yields as good a solution in 1D cases as and better solutions in 2D than the one using the first more elaborate flux limiter. The scheme also preserves monotonicity in the solutions and is computational much more efficient.

The simple flux-limited fourth- and sixth-order diffusion schemes are also applied to thermal bubble convection. It is shown that overshooting and undershooting are consistently smaller when the flux-limited version of the high-order diffusion is used, no matter whether the advection scheme is monotonic or not. This conclusion applies to both scalar and momentum fields. Higher-order monotonic diffusion works better and even more so when used together with monotonic advection.

1. Introduction

Most numerical models employ numerical diffusion or computational mixing to control small-scale (near two grid intervals in wavelength) noise that can arise from numerical dispersion, nonlinear instability, discontinuous physical processes, and external forcing. Such diffusion cannot always be substituted for by physresents other processes or sources. The last term on the right-hand side (rhs) is the added diffusion term and n $(=0, 2, 4, 6, \ldots)$ denotes the order. Here α_n is called diffusion coefficient. Diffusion or smoothing can also be introduced by periodically applying a spatial filter or smoother to the predicted field (e.g., Shapiro 1970, 1975). Its effect is often equivalent to applying diffusion in the prognostic equation for a period of time.

Discretization

Simplifications

- Ignore other processes (S=0)
- 4th-order (n=4)
- Without limiter (see paper)

$$\frac{\partial \phi}{\partial t} = S + (-1)^{n/2+1} \alpha_n \nabla^n \phi, \qquad (1)$$

$$S = 0 \quad n = 4 \quad \text{horizontal}$$

$$\begin{aligned} \frac{\partial \phi}{\partial t} &= -\alpha_4 \nabla_h^4 \phi \\ &= -\alpha_4 \Delta_h^2 \phi \qquad \Delta_h = \nabla_h^2 \\ &= -\alpha_4 \Delta_h \left(\Delta_h \phi \right) \end{aligned}$$

Spatial disretization

2nd-order central finite differences

$$\Delta_h \phi_{i,j}^n \approx \left(-4\, \phi_{i,j}^n + \phi_{i-1,j}^n + \phi_{i+1,j}^n + \phi_{i,j-1}^n + \phi_{i,j+1}^n \right) / \Delta x^2$$

Time discretization

1st-order forward (Euler)

$$\partial_t \phi_{i,j}^n \approx \left(\phi_{i,j}^{n+1} - \phi_{i,j}^n \right) / \Delta t$$

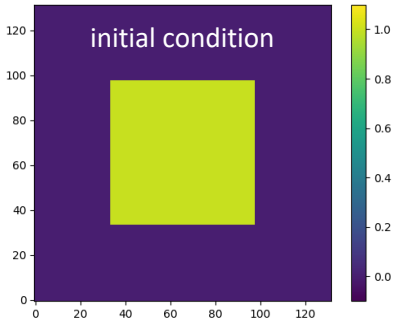
Structure of stencil_2d.F90 / stencil_2d.py

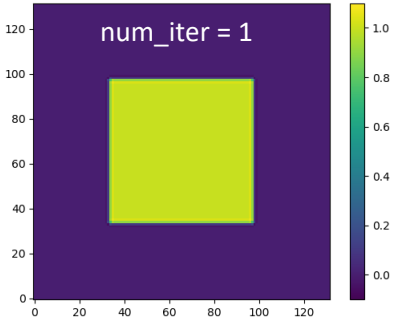
$$\mathtt{tmp}_{\mathrm{i,j,k}} = lap(\mathtt{in}_{i,j,k}) = -4\,\mathtt{in}_{i,j,k} + \mathtt{in}_{i-1,j,k} + \mathtt{in}_{i+1,j,k} + \mathtt{in}_{i,j-1,k} + \mathtt{in}_{i,j+1,k}$$

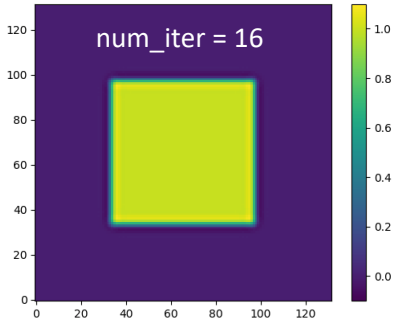
$$\mathtt{out}_{\mathrm{i,j,k}} = lap(\mathtt{tmp}_{i,j,k}) = -4\,\mathtt{tmp}_{i,j,k} + \mathtt{tmp}_{i-1,j,k} + \mathtt{tmp}_{i+1,j,k} + \mathtt{tmp}_{i,j-1,k} + \mathtt{tmp}_{i,j+1,k}$$

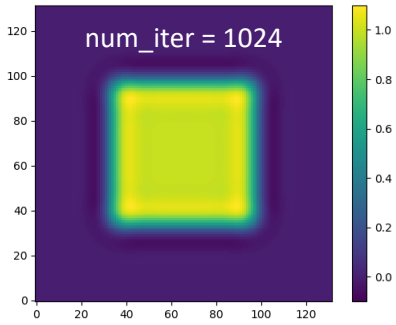
$$\mathtt{out}_{\mathrm{i,j,k}} = \mathtt{in}_{\mathrm{i,j,k}} - \alpha\ \mathtt{out}_{\mathrm{i,j,k}}$$

$$\texttt{if iter} < \texttt{num\_iter} \ : \ \mathrm{swap}\,(\mathtt{in}, \mathtt{out})$$









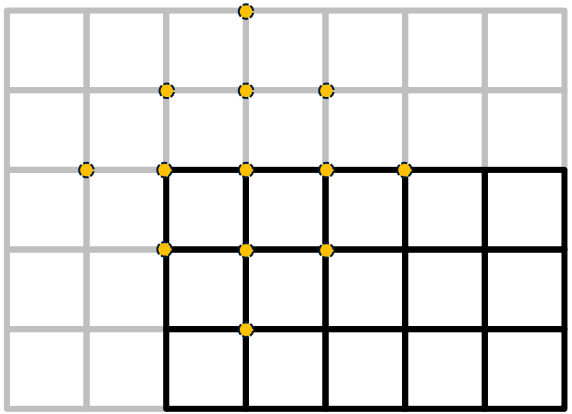
Halo points

halo_update(in)

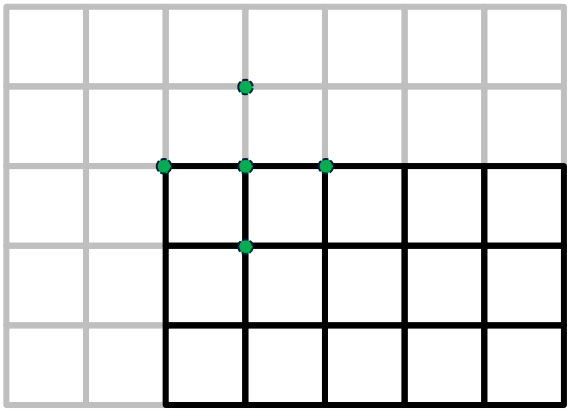
$$\mathtt{tmp}_{i,j,k} = lap(\mathtt{in}_{i,j,k}) = -4\,\mathtt{in}_{i,j,k} + \mathtt{in}_{i-1,j,k} + \mathtt{in}_{i+1,j,k} + \mathtt{in}_{i,j-1,k} + \mathtt{in}_{i,j+1,k}$$

$$\mathtt{out}_{i,j,k} = lap(\mathtt{tmp}_{i,j,k}) = -4\,\mathtt{tmp}_{i,j,k} + \mathtt{tmp}_{i-1,j,k} + \mathtt{tmp}_{i+1,j,k} + \mathtt{tmp}_{i,j-1,k} + \mathtt{tmp}_{i,j+1,k}$$

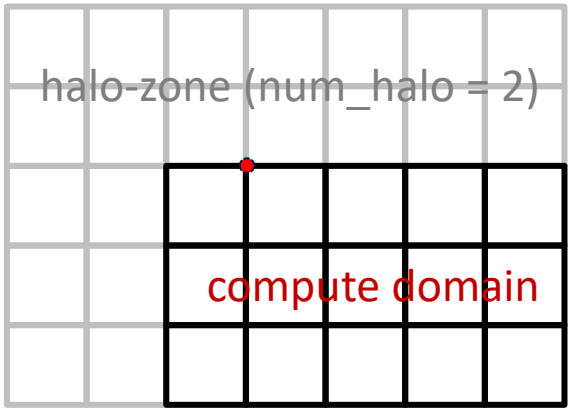
$$\mathtt{out}_{i,j,k} = \mathtt{in}_{i,j,k} - \alpha\,\mathtt{out}_{i,j,k}$$



in(i,j,1)

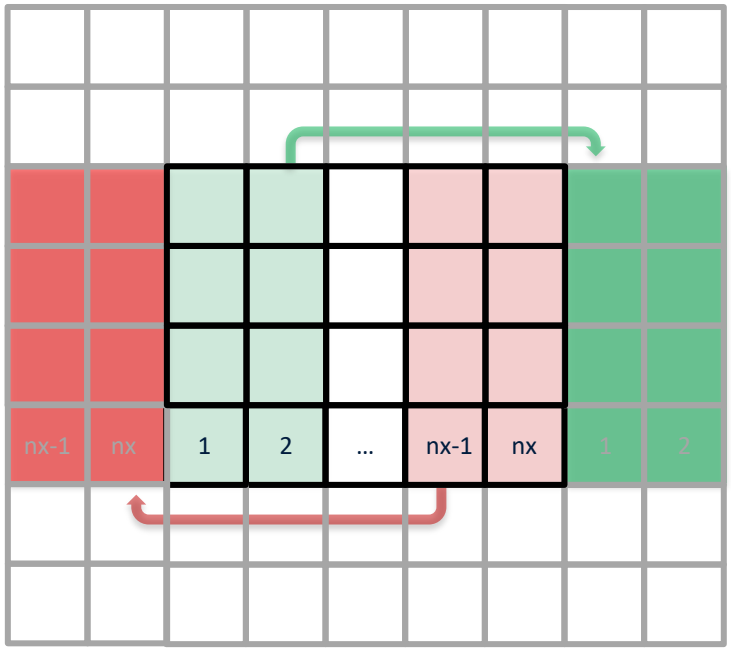


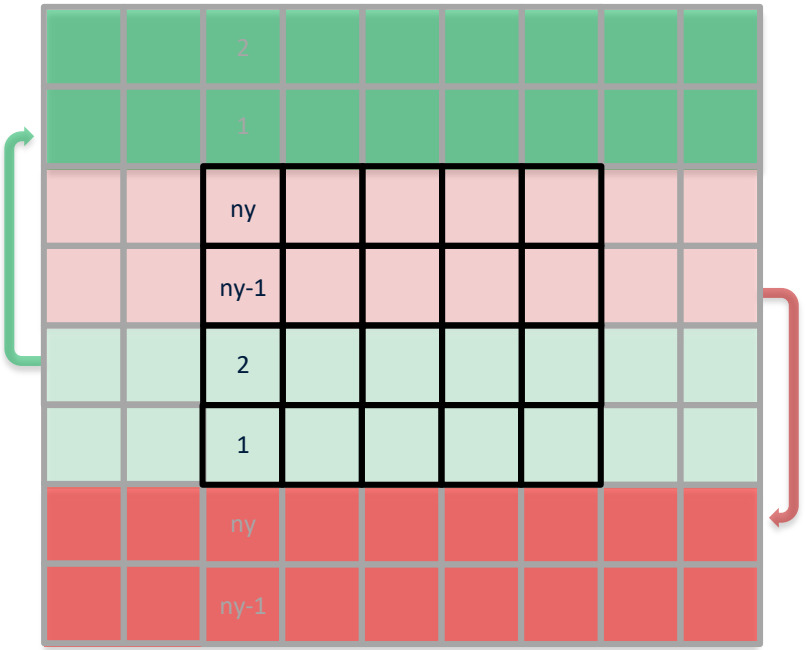
tmp(i,j,1)



out(i,j,1)

Halo update (periodic)






```
field(nx + 2 * num_halo, ny + 2 * num_halo, nz)
```