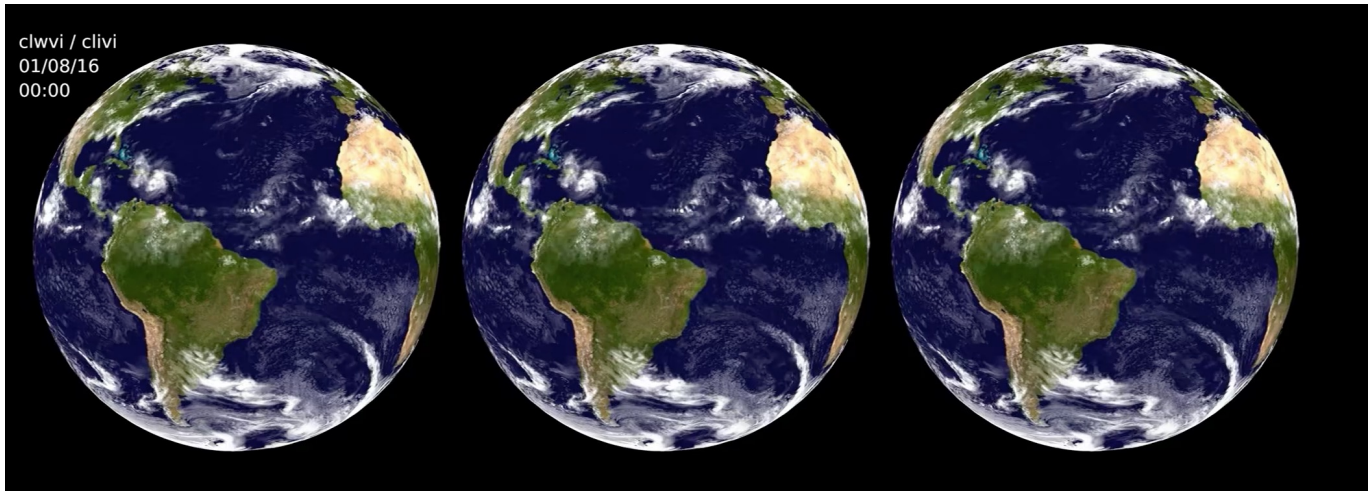
Met Office | NATURAL ENVIRONMENT RESEARCH COUNCIL

Joint Weather & Climate Research Programme

A partnership in climate research

Weather and Climate modelling: ready for exascale?



National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

P.L. Vidale

Department of Meteorology
University of Reading, UK

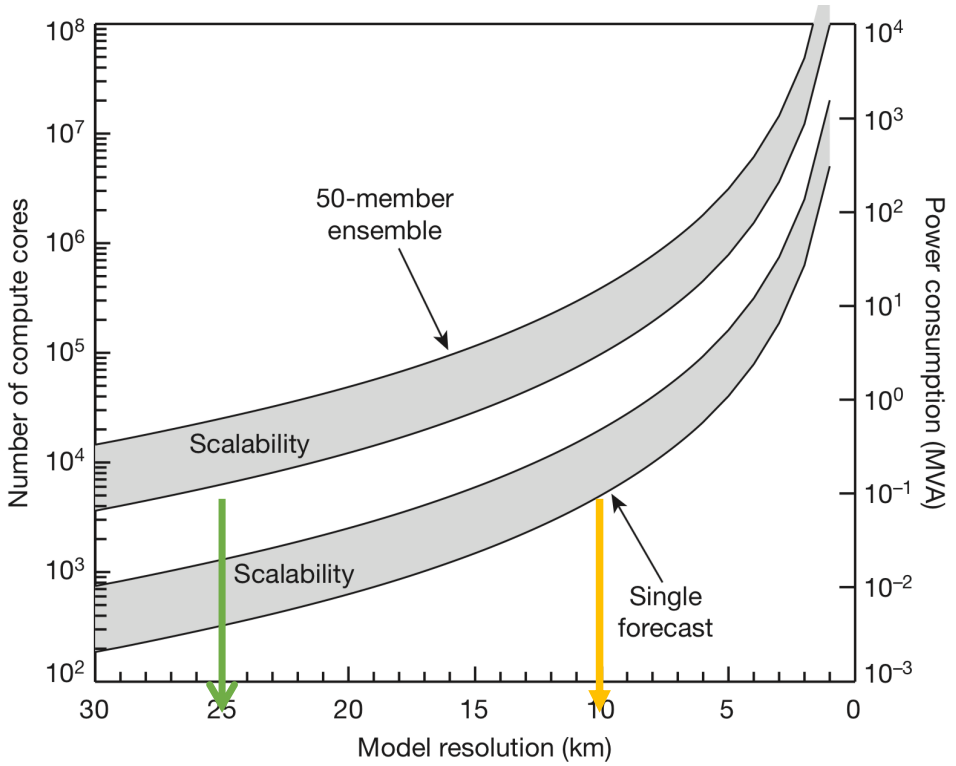


Figure 5 | CPU and power requirements as a function of NWP model resolution. Simplified illustration of the number of compute cores (left *y*-axis)

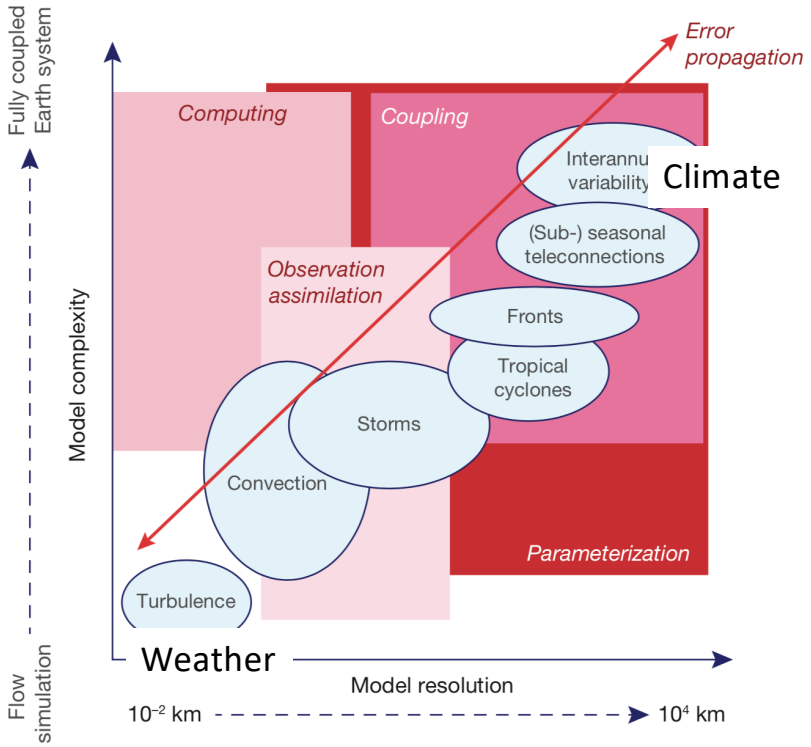


Figure 6 | Key challenge areas for NWP in the future. Advances in forecast

For NWP centres such as ECMWF, the upper limit for affordable power usage may be about 20 MVA (ref. 91). The likely future NWP system will be of the order of 100–1,000 times larger as a computational task than today's systems, and would require about 10 times more power. Figure 5 illustrates the increase in compute cores and electric power supply if model resolution is increased for a single forecast and a 50-member ensemble, assuming today's model design and available technology. To approach the resolutions of 1–5 km that are considered crucial for resolving convection, high-performance computers of unpre- cedented dimension and cost (assuming the use of conventional technology) would be required.

Bauer et al. Nature 2015

A "prediction" made in 2008
Model Grid Size (km) & Computing Capability

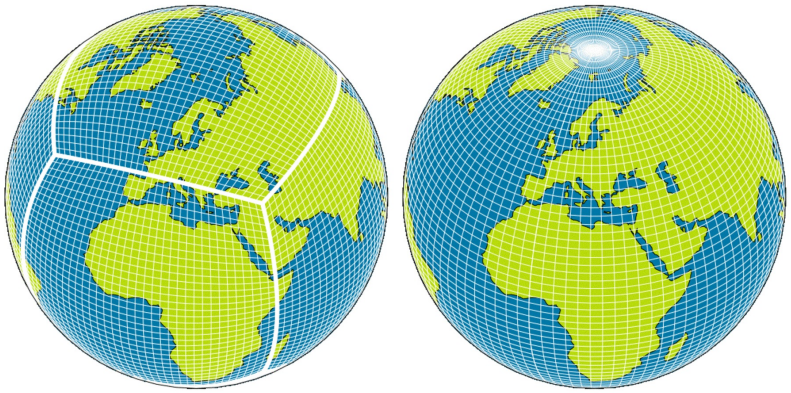
	Earth Simulator 2002-2009		PRACE-HERMIT 2012-		
Peak Rate:	**10 TFLOPS**	**100 TFLOPS**	**1 PFLOPS**	**10 PFLOPS**	**100 PFLOPS**
Cores	1,400 (2005)	12,000 (2007)	80-100,000 (2009)	300-800,000 (2011)	6,000,000? (20xx?)
Globa 5-1					1.4
S 50-100					1.3
I 5-1 yr/day					4.2
Climate Change[2]:	120 - 200	57 - 91	27 - 42	12 - 20	5.7 - 9.1

* Core counts above $O(10^4)$ are unprecedented for weather or climate codes, so the last 3 columns require getting 3 orders of magnitude in scalable parallelization

2022 fact check:

- ECMWF's new Atos machine is 30 PF with 1 million cores
- Will be running 50 ensemble members at 9km resolution, we assume >1PF sustained
- Highres we use ~51000 cores for 4 days / hour at 2.5 km resolution

State-of-the-art supercomputers suitable for weather prediction (NWP), 2018-



Numerical weather prediction models solve the equations of motion globally on a grid of only 10 x 10 km on 90 levels (N1280, at MO)

→ 4.2E8 grid points

Time-step 5 mins → **1.2E11 computations per day**

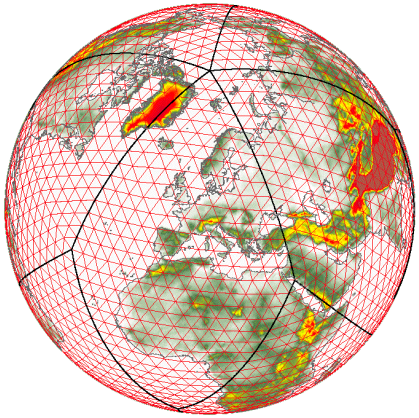
ECMWF has been running 9km NWP, with 50 ensemble members, then "deterministic" at 2.5km

Nils Wedi has run 1.4km on CSCS Piz Daint (4888 nodes) and DOE's Summit (3840 nodes)

For the future, we also increasingly aim to couple with the ocean

Latest supercomputers have already breached the so-called PetaScale:

1. Archer-2 (EPSR/NERC) is rated at nearly at 28 PetaFlops
2. New ECMWF machine in Italy rated at 30 PetaFlops → **going to 100 member ensembles**
3. The current (old) MO supercomputer is rated at nearly 16 PetaFlops



Evolution of technology for weather and climate science

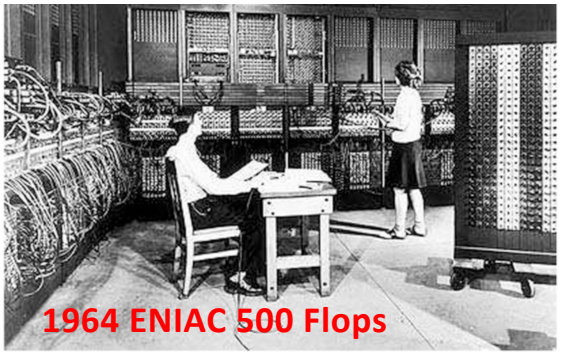
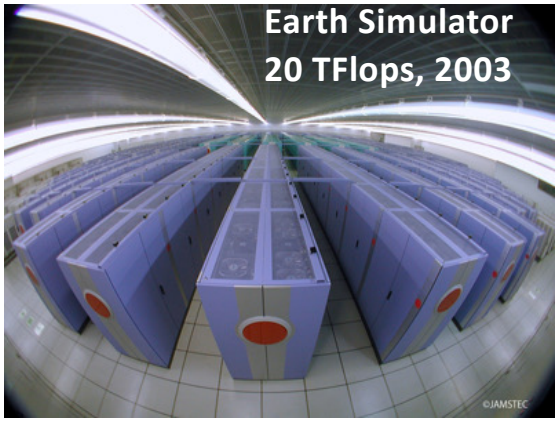
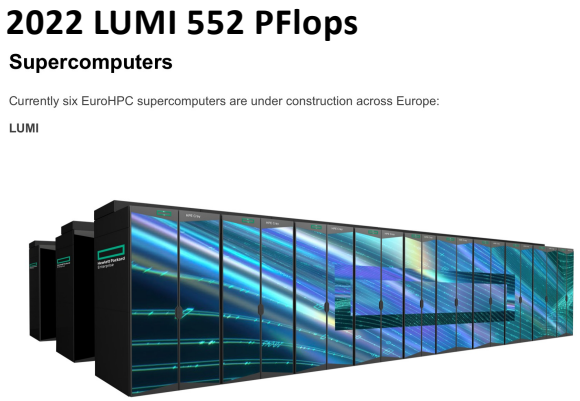


FIG. 12-2. The Electronic Numerical Integrator and Computer (ENIAC). [Courtesy of International Business Machines Corporation, ©1946 International Business Machines Corporation.]





1. Single-core, clock frequency used to increase each year (Moore's law)
2. Vector processors: do far more with each clock cycle
3. Multi-core: distribute the work among a large number of processing units
4. Hybrid (e.g. CPU and GPU): use different types of processors to perform specialised tasks

EuroHPC: €8 billion programme to take us towards exascale

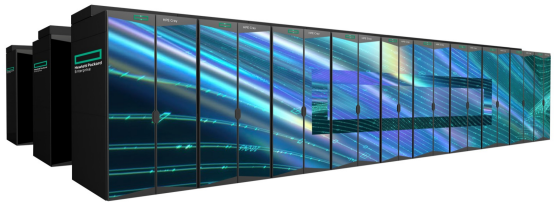
- 3 large (O(100PFlops)) supercomputers in Finland, Italy, Spain
- 5 smaller ones (size of Archer) in Luxembourg, Slovenia, Portugal, Czech Republic, Bulgaria

Supercomputers

Currently six EuroHPC supercomputers are under construction across Europe:

LUMI

Finland, 552/375 Pflops



© *HPE*

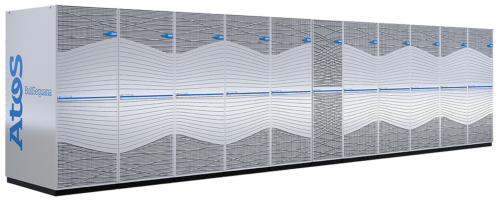
The LUMI system will be a Cray EX supercomputer supplied by Hewlett Packard Enterprise (HPE) and located in Finland.

Sustained performance:	375 petaflops
Peak performance:	552 petaflops
Compute partitions:	GPU partition (LUMI-G), x86 CPU-partition (LUMI-C), data analytics partition (LUMI-D), container cloud partition (LUMI-K)
Central Processing Unit (CPU):	The LUMI-C partition will feature 64-core next-generation AMD EPYC™ CPUs
Graphics Processing Unit (GPU):	LUMI-G based on the future generation AMD Instinct™ GPU
Storage capacity:	LUMI's storage system will consist of three components. First, there will be a 7-petabyte partition of ultra-fast flash storage, combined with a more traditional 80-petabyte capacity storage, based on the Lustre parallel filesystem, as well as a data management service, based on Ceph and being 30 petabytes in volume. In total, LUMI will have a storage of 117 petabytes and a maximum I/O bandwidth of 2 terabytes per second
Applications:	AI, especially deep learning, and traditional large scale simulations combined with massive scale data analytics in solving one research problem
Other details:	LUMI takes over 150m2 of space, which is about the size of a tennis court. The weight of the system is nearly 150 000 kilograms (150 metric tons)

The LUMI consortium includes the Swiss CSCS, which currently owns one of the world's top 10 supercomputers

LEONARDO

Italy: 322/249 PFlops



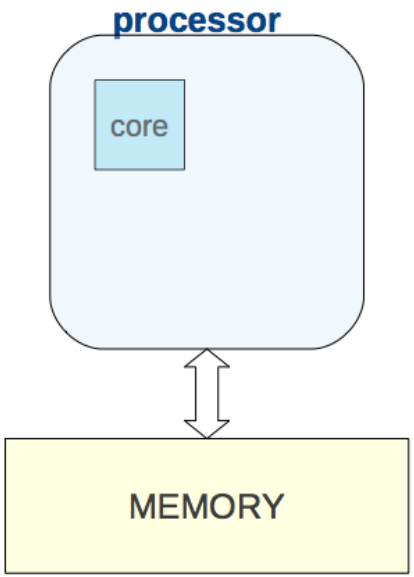
© *Atos*

LEONARDO will be supplied by ATOS, based on a BullSequana XH2000 supercomputer and located in Italy.

Sustained performance:	249.4 petaflops
ak formance:	322.6 petaflops
mpute titions:	Booster, hybrid CPU-GPU module delivering 240 PFlops, Data-Centric, delivering 9 Pflops and featuring DDR5 Memory and local NVM for data analysis
ntral ocessing it (CPU):	Intel Ice-Lake (Booster), Intel Sapphire Rapids (data-centric)
aphics ocessing it (GPU):	NVIDIA Ampere architecture-based GPUs, delivering 10 exaflops of FP16 Tensor Flow AI performance
rage acity :	Leonardo is equipped with over 100 petabytes of state-of-the-art storage capacity and 5PB of High Performance storage
plications:	The system targets: modular computing, scalable computing applications, data-analysis computing applications, visualization applications and interactive computing applications, urgent and cloud computing
er details:	Leonardo will be hosted in the premises of the Tecnopolo di Bologna. The area devoted to the EuroHPC Leonardo system includes 890 sqm of data hall, 350 sqm of data storage, electrical and cooling and ventilation systems, offices and ancillary spaces

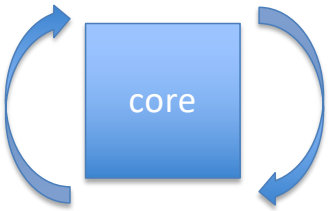
Single core scalar and vector processing

Older processor had only one cpu core to execute instructions



real, dimension(N) :: V1,V2,V3

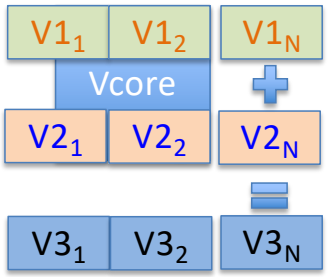
V3=V1+V2



Scalar calculation: single sum per clock cycle

REPEAT N times

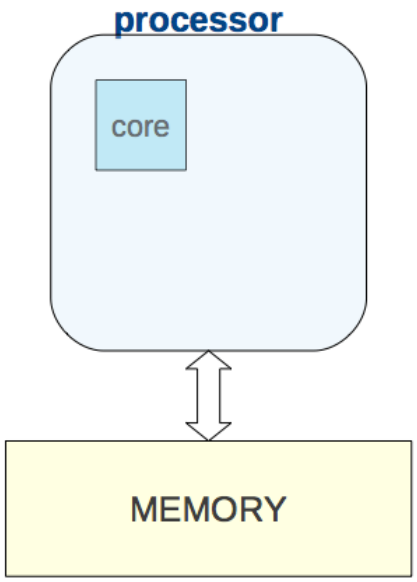
V3=V1+V2



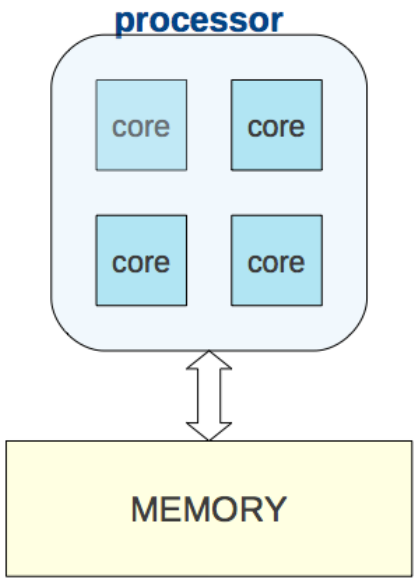
Vector calculation: entire sum in a single clock cycle

Going to multi-core

Older processor had only one cpu core to execute instructions

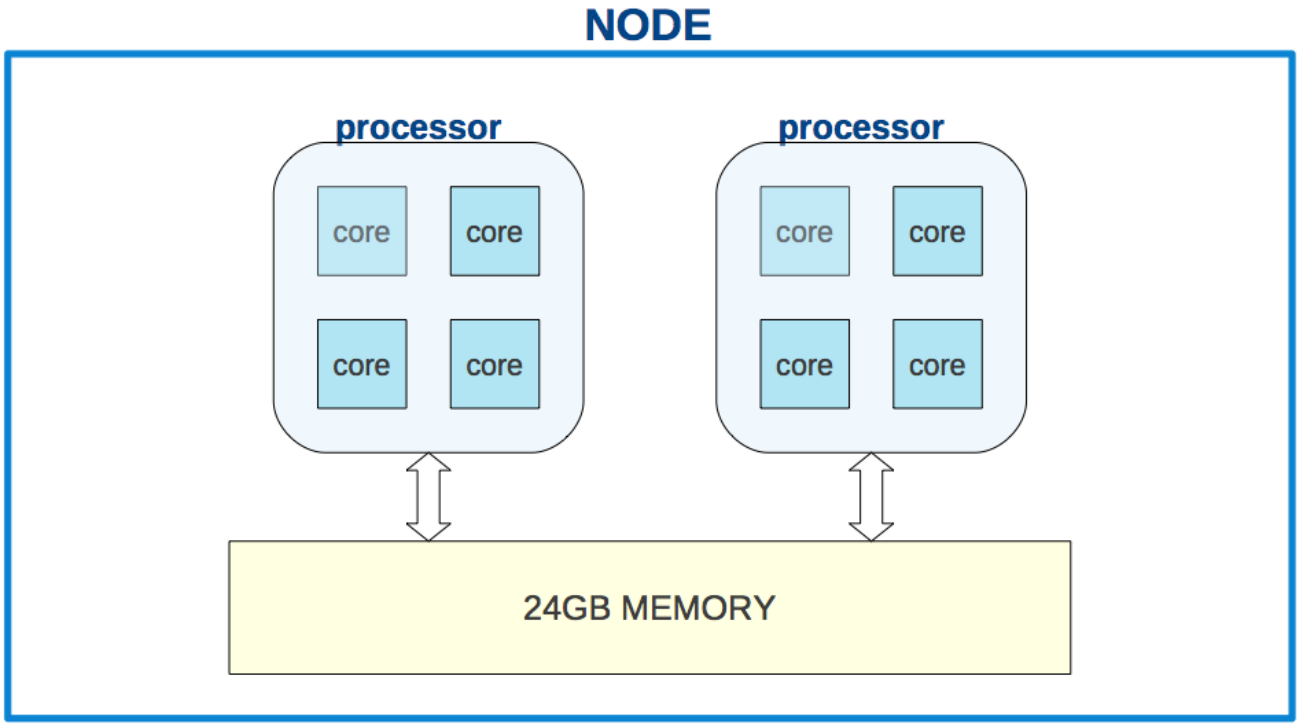


Modern processors have 4 or more independent cpu cores to execute instructions



And parallel across processor nodes

All cores see the entire domain over a shared memory. Fast memory access and no need for communication. But this cannot scale to a large number of cores.



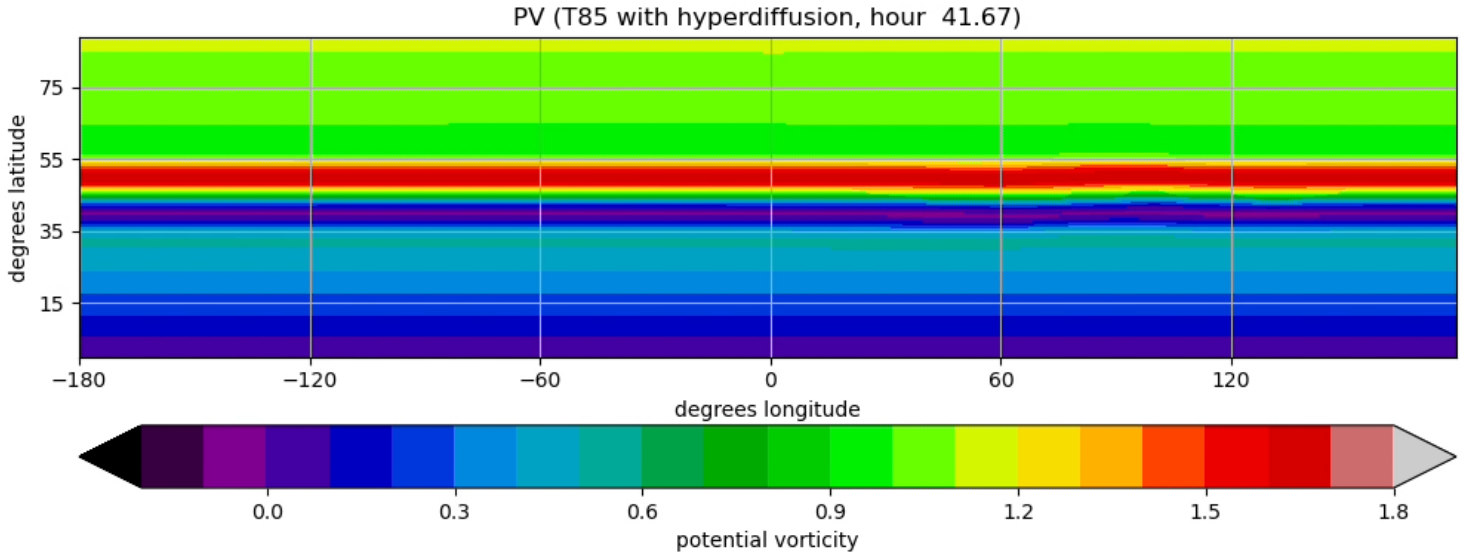
Testing all these technologies on a simple problem

Matrix multiplication: C=AB

What is the performance on this Mac if arrays are: ***real, dimension (1024,1024) :: A,B***

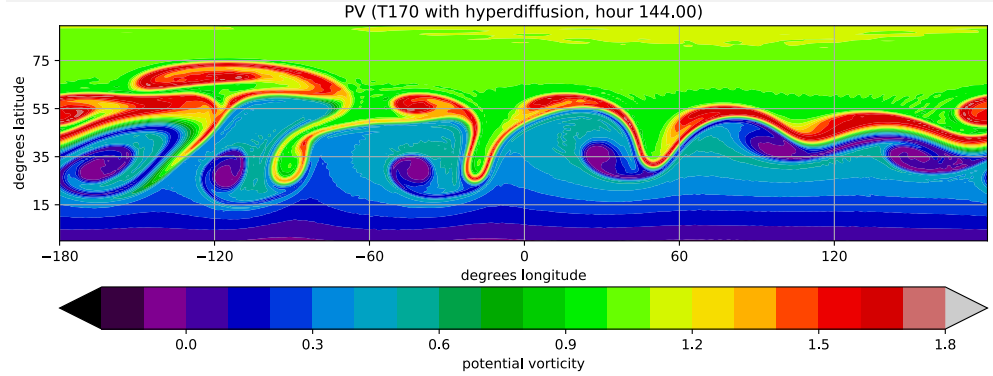
FORTRAN90	Elapsed (s)	Speedup	Comments
scalar	0.031		Very hard to disable automatic vectorization
vector	0.002	15	Short vectors on Intel processors!
Parallel (shared memory) OpenMP	0.003-0.0006	10-50	1-16 threads Overhead to be paid, and limited scalability
Parallel (message-passing) MPI			Too much trouble for such a small problem. See NUMPY example

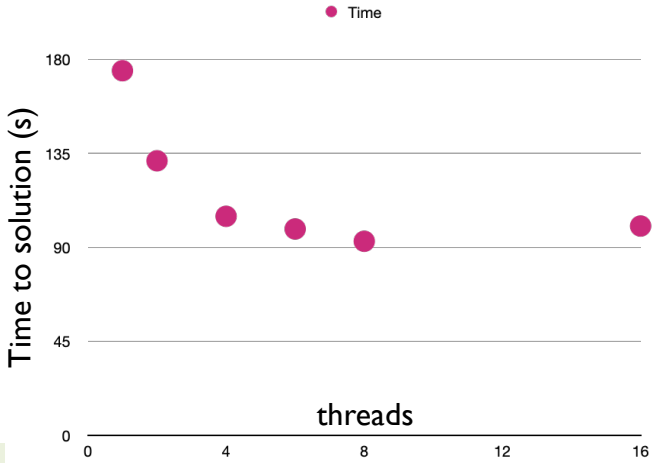
Galewsky's barotropic instability problem with a global spectral model (T85)



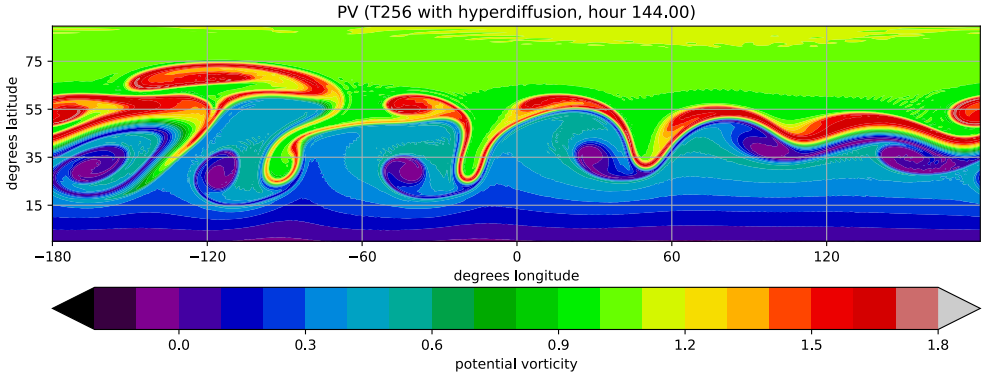
SHTNs: an efficient library enabling multi-threading and use of GPU

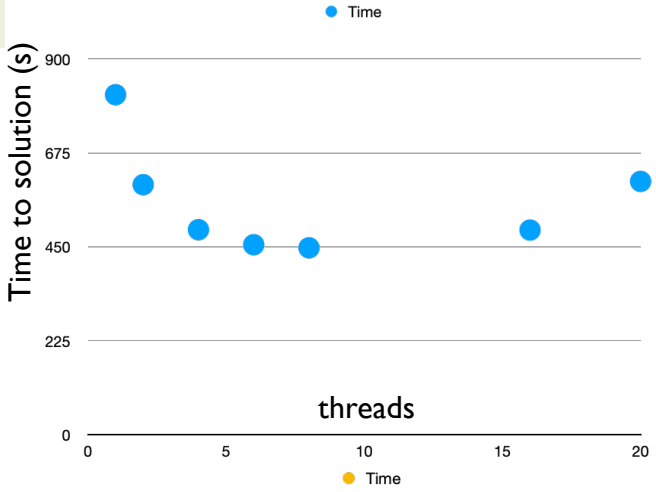
T170: small-size problem, little IO and use of swapping to disk



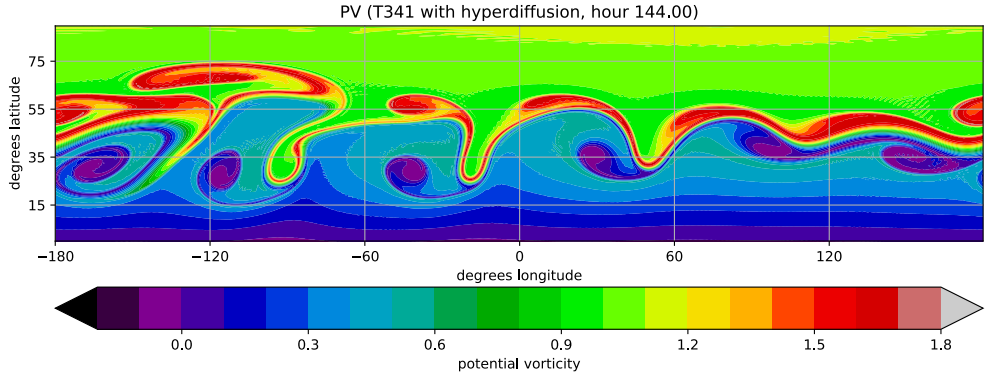


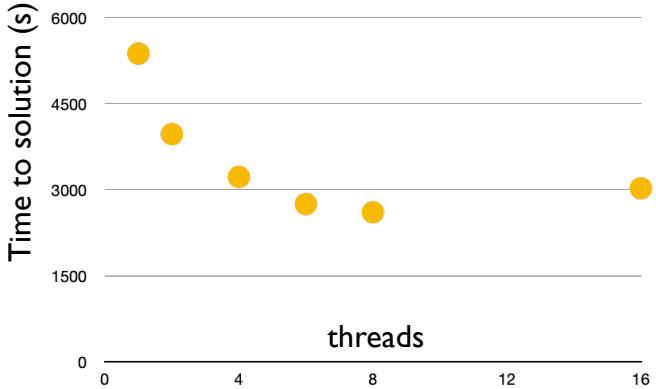
T256: medium-size problem, 95% of CPU used by computation, little IO and use of swapping to disk





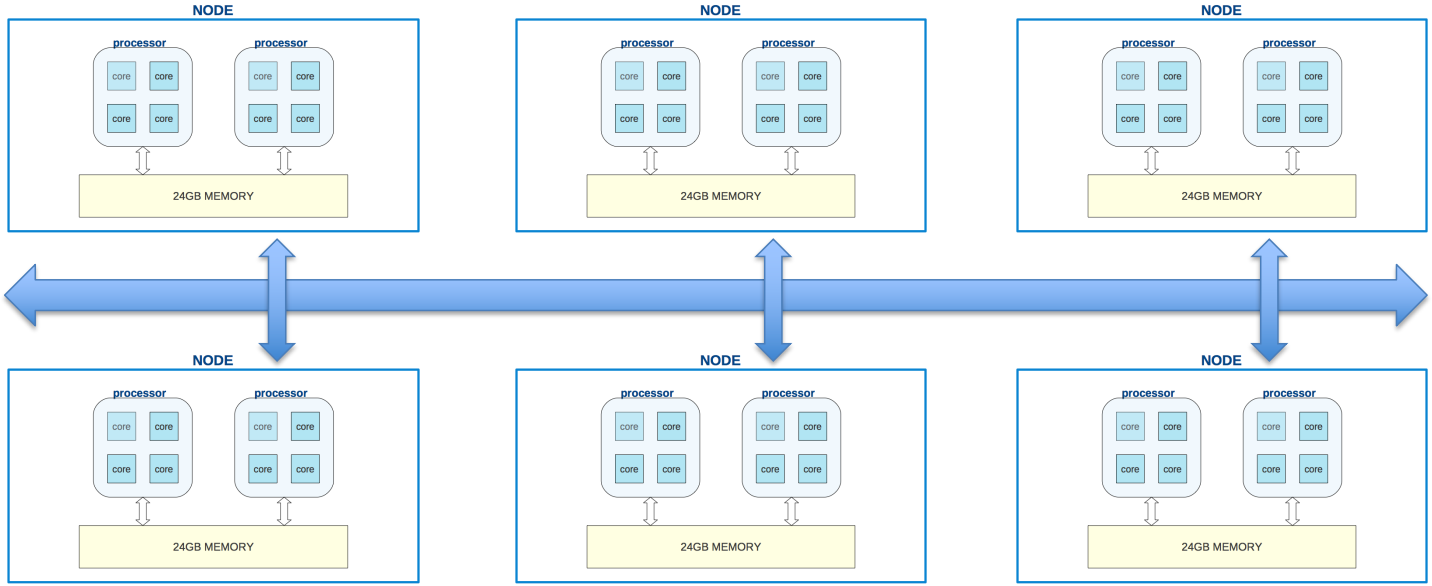
T341: large-ish problem, Only 15% of CPU used by computation, 85% used by system: IO and use of swapping to disk



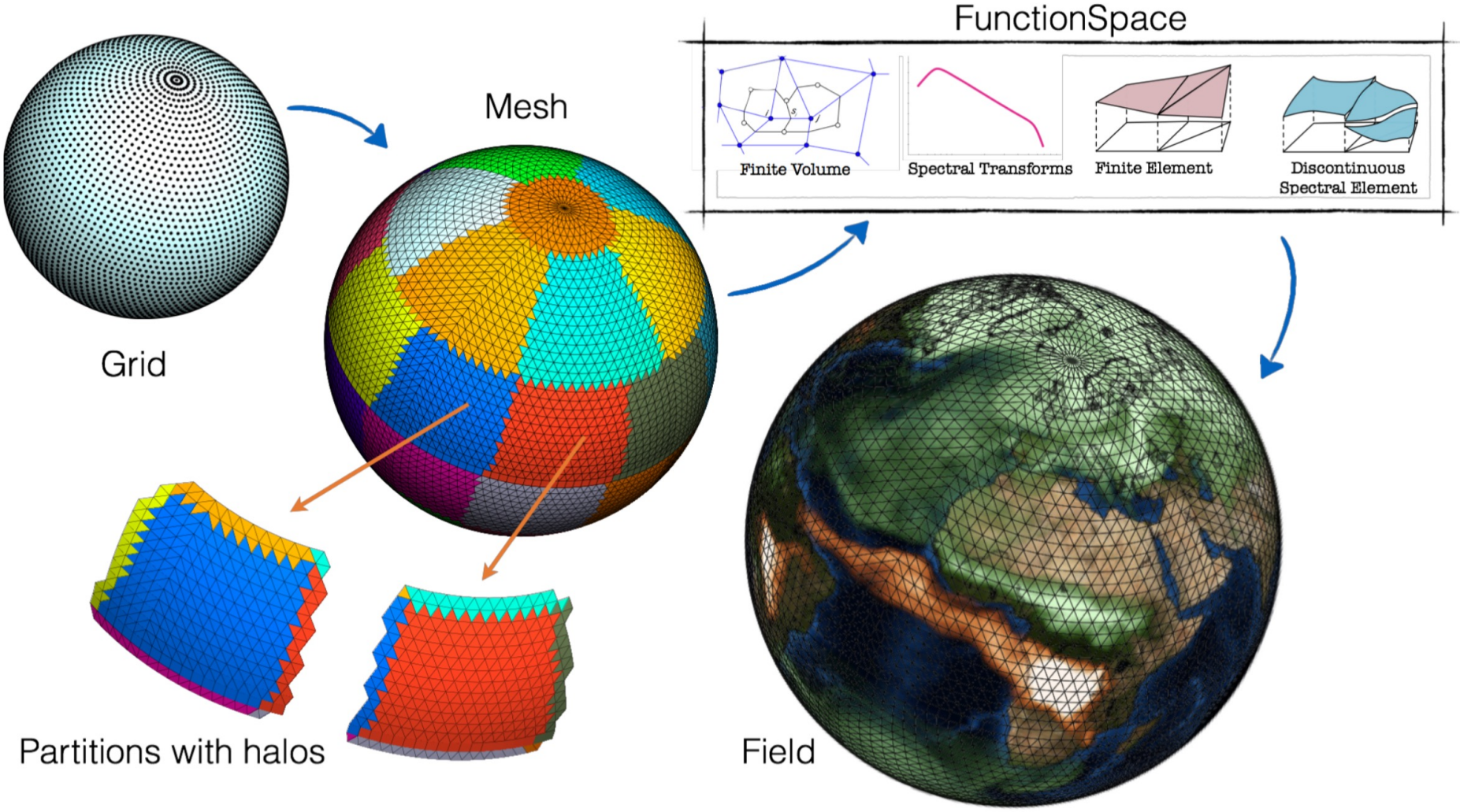


And across cabinets

Message-passing across cabinets over fast networks. Memory is not shared and typical domain decomposition must include “halos” so that each node can access a surrogate of boundary conditions, as it can only see a limited region, not the entire domain.

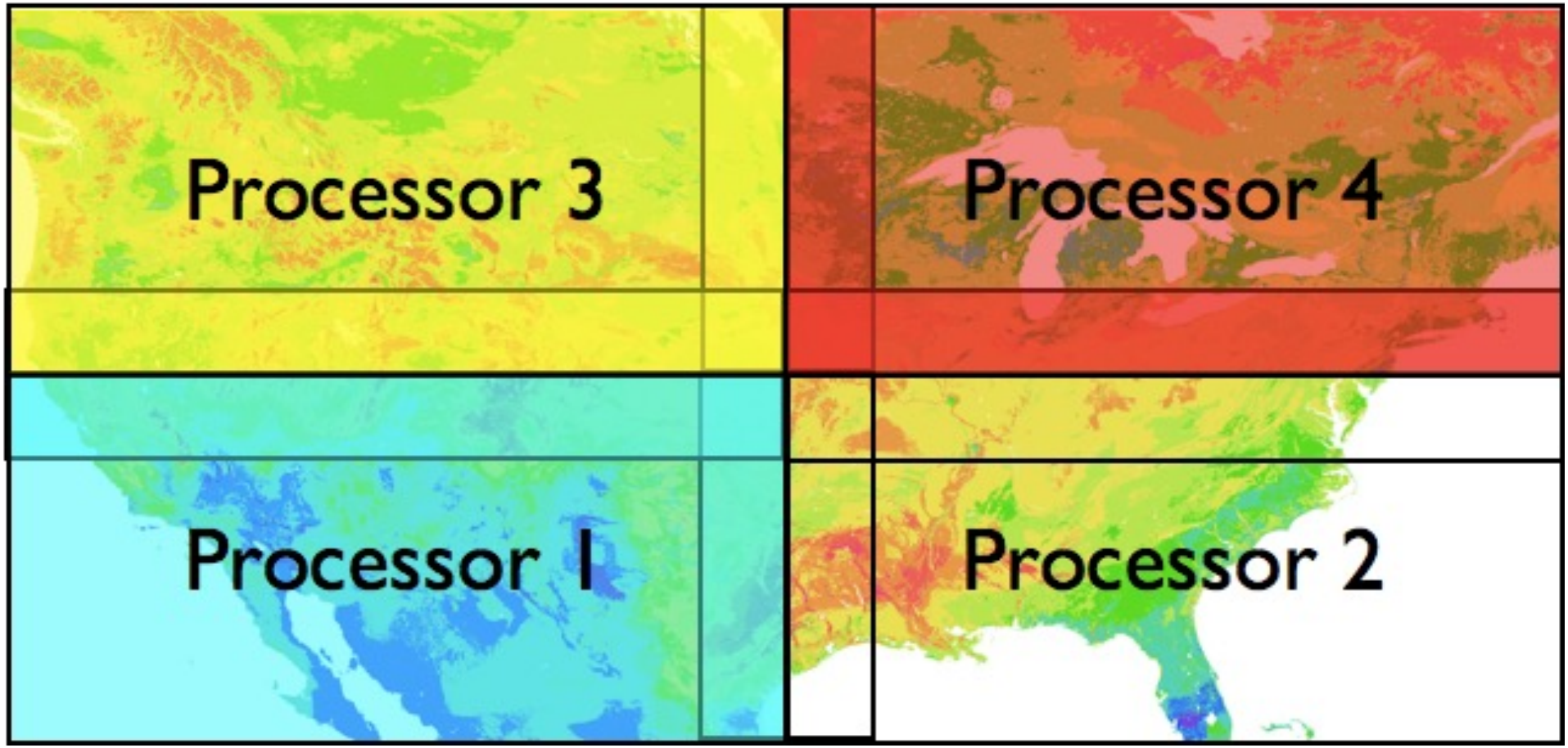


Our type of model issues a very large number of communications. This is very inefficient.



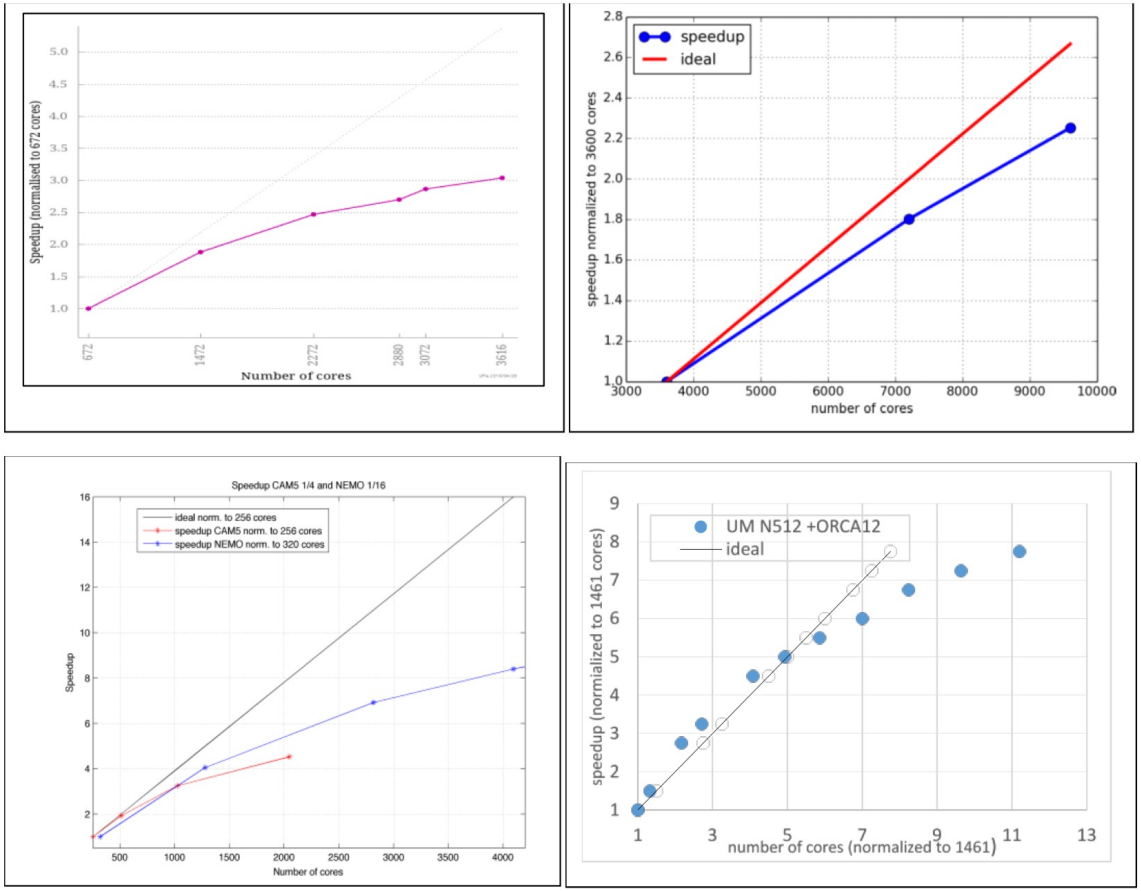
Deconinck et al. 2017

Parallel computing and haloes



Portion of a global domain treated as a collage of limited area models.
The main purpose of the halo regions is to reduce the requirement for frequent communications between computer nodes, which are very slow.
Can you see a problem with the use of the semi-Lagrangian method?

Current performance and scalability of some state-of-the-art GCMs



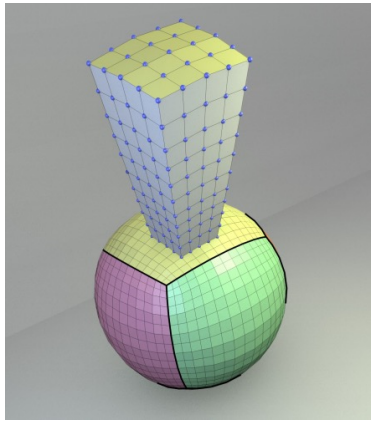
Model	Atmospheric resolution	Oceanic resolution
HadGEM3-NEMO	25km	1/12°(ORCA12 L85)
ECHAM-FESOM	50km (T255L95)	Variable, ¼ to 1/12°
CAM-NEMO	25km	¼ ° -1/16°(ORCA16 L98)
ECEarth-NEMO	25km (T511L91)	¼° (ORCA025 L75)

Figure 1: the scaling characteristics of the four HiPRACE models. a) EC-Earth (top left); b) AWI ECHAM-FESOM (top right); c) CMCC (bottom left) and d) MO-UM (bottom right).

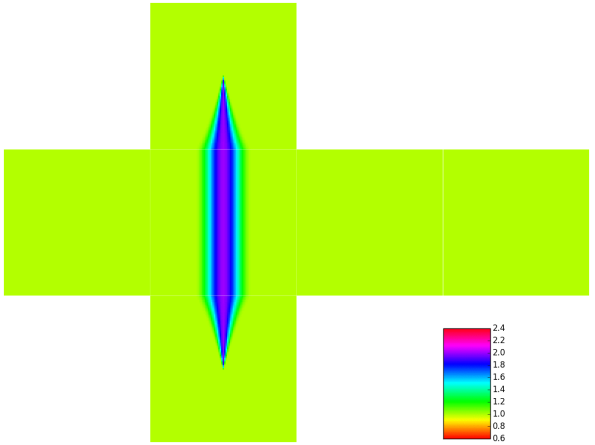
Run type	# Runs	# Steps/Run	Walltime/Step [hours]	# CPU cores	Total core hours/Type Run
EC-Earth: MareNostrum	6 HR	100	8.2	2500	**13,530,000**
AWI: Curie	3+1[3] VHR	100	9.6	14,208	**52,512,768**
CMCC: MareNostrum	3 VHR	1200	4.7	6144	**149,681,664**
	3 HR	1200	4.1	2176	
MO-UoR: Hazel Hen	3 VHR	100	22.9	7206	**54,353,828**

GungHo: the design

- Quadratic mesh
  - Cubed sphere/diamond mesh



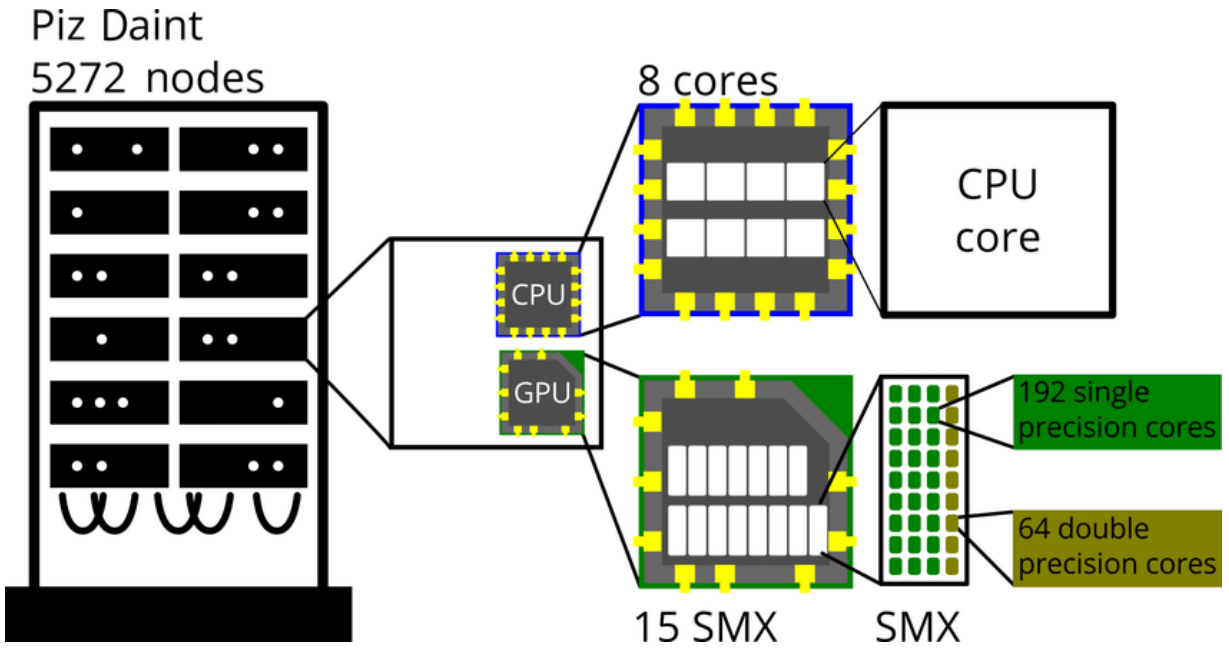
- Mixed finite elements (see next slide)
- Semi-implicit
- Conservative transport



Exploiting parallel computing to speed up prediction

Unlike the computers we used to have in the past (scalar, then vector), we currently make use of parallel computers.

Basically, similar to what is inside your PC or Mac, with multi-core chips, we combine multiple processors on a board, and multiple boards inside a cabinet.



Piz Daint currently at 5MW
Top US supercomputer (Frontier) at 25-30MW

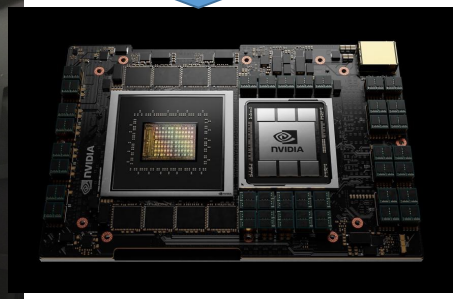
Managing O(10)MW of power for these machines is a challenge

Piz Daint to Alps

Specifications

Model	Cray XC40/XC50
XC50 Compute Nodes	Intel® Xeon® E5-2690 v3 @ 2.60GHz (12 cores, 64GB RAM) and NVIDIA® Tesla® P100 16GB - 5704 Nodes
XC40 Compute Nodes	Two Intel® Xeon® E5-2695 v4 @ 2.10GHz (2 x 18 cores, 64/128 GB RAM) - 1813 Nodes
Login Nodes	Intel® Xeon® CPU E5-2650 v3 @ 2.30GHz (10 cores, 256 GB RAM)
Interconnect Configuration	Aries routing and communications ASIC, and Dragonfly network topology
Scratch capacity	8.8 PB

CSCS is moving from Intel to ARM architectures for its CPUs, and better integration between CPUs and GPUs Grace Hopper in Alps



Users who do not move to GPU will find that their applications will not speed up

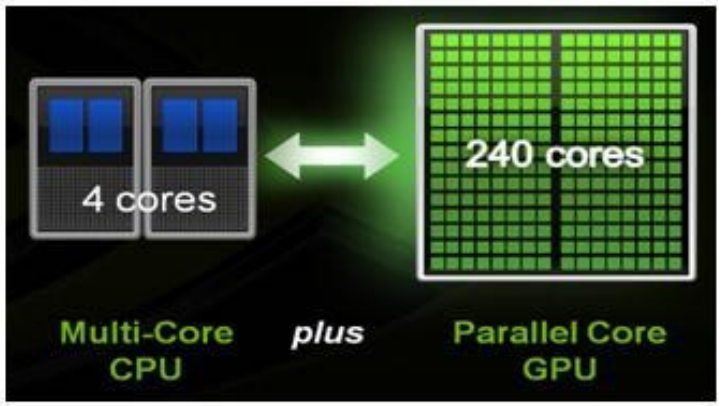
Testing all these technologies on a simple problem

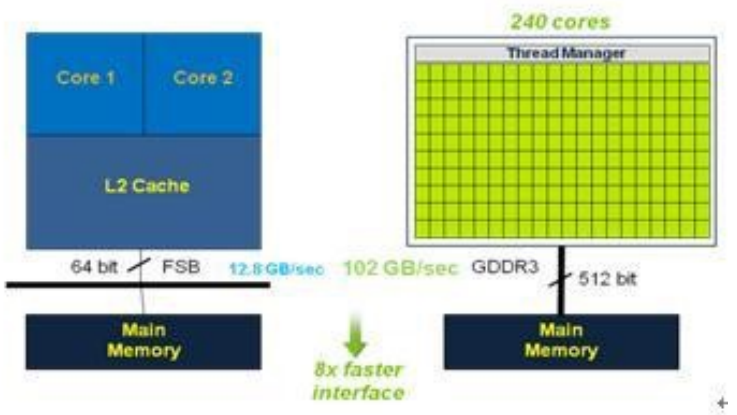
Matrix multiplication: C=AB

Same problem with Python, NUMPY, NUMBA, CUDA

Technology	Elapsed (s)	Speedup	Comments
F90 scalar	0.031		Very hard to disable automatic vectorization
NUMPY dot	0.014	~2	
NUMPY explicit	0.001	31	
CUDA, no shared memory	0.787	0.04	
CUDA, shared memory	0.284	0.1	
CUDA-BLAS	0.01	3.1	

What's CUDA?





NVIDIA® CUDA® is a parallel computing platform and programming model that

- enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU)
- works as extension of many common languages
- Example of what is found in modern hybrid supercomputers

Were GPUs not supposed to be the future?

1. Need to configure the problem in a way that is appetising to the GPU

Technology	Elapsed (s)	Speedup	NUMPY (CPU)
CUDA, shared memory	0.284		
CUDA, shared memory and Explicit 64*128*128	0.32	0.88	0.08
CUDA, shared memory and Explicit 262144*2*2	0.001	284	0.0015

GPUs are not magic and there is a steep price to pay:

1. you need to know what you are doing in order to obtain a speedup.
2. you need to convert arrays to special CUDA arrays, because the GPU number representation is different from that on the CPU
3. There is a cost in moving arrays from CPU to GPU memory and back again

A good algorithm (e.g. NUMPY) can give you what you need without the extra complication.

But GPUs do use less power!!

Lessons from Petascale Weather and Climate simulation projects in 2013-2018

PRACE-UPSCALE (2012-2013)
One AGCM at 25km, 8 ensemble members, 27 years of simulation each
- 144 million core hours
- 600 TB written to disk

H2020 PRIMAVERA (2016-2020)
CMIP6 (HigResMIP)
6 GCMs at ~20km, 1 ensemble member

100 years of simulation, 2 experiments per group

	TOT
HPC (core hours)	503 million
Written to disk	~13.5 PB
Long-term storage	~9.1PB
Energy costs (GWhr)	1.57 *

* About 460 households for one year

If we were to operate in the same way, in 2024, at 1km, CMIP7 would output 80 exabytes of data.

Things will become much much harder in the future
Estimates by Oliver Fuhrer and colleagues

For a 1-year global climate simulation:

A modern simulation's costs are O(MW) hours per simulated year (MWh/SY)
Piz Daintz simulation power draw: 2MW; TaihuLight simulation: 15MW
ECMWF max achievable capability in the future: 10-13MW

A conventional **"cloud-resolving" simulation** would cost about 3000 MWh/SY, nearly the amount of energy used by 900 households in one year.

A novel, energy-efficient **cloud-resolving simulation (DX=930m)** with COSMO costs about 596 MWh/SY **
For the European Centre's IFS, a similar simulation requires 192 MWh/SY

$\langle \Delta x \rangle$	#nodes	Δt [s]	SYPD	MWh/SY	gridpoints
930 m	4888	6	0.043	596	3.46×10^{10}
1.9 km	4888	12	0.23	97.8	8.64×10^{9}
47 km	18	300	9.6	0.099	1.39×10^{7}

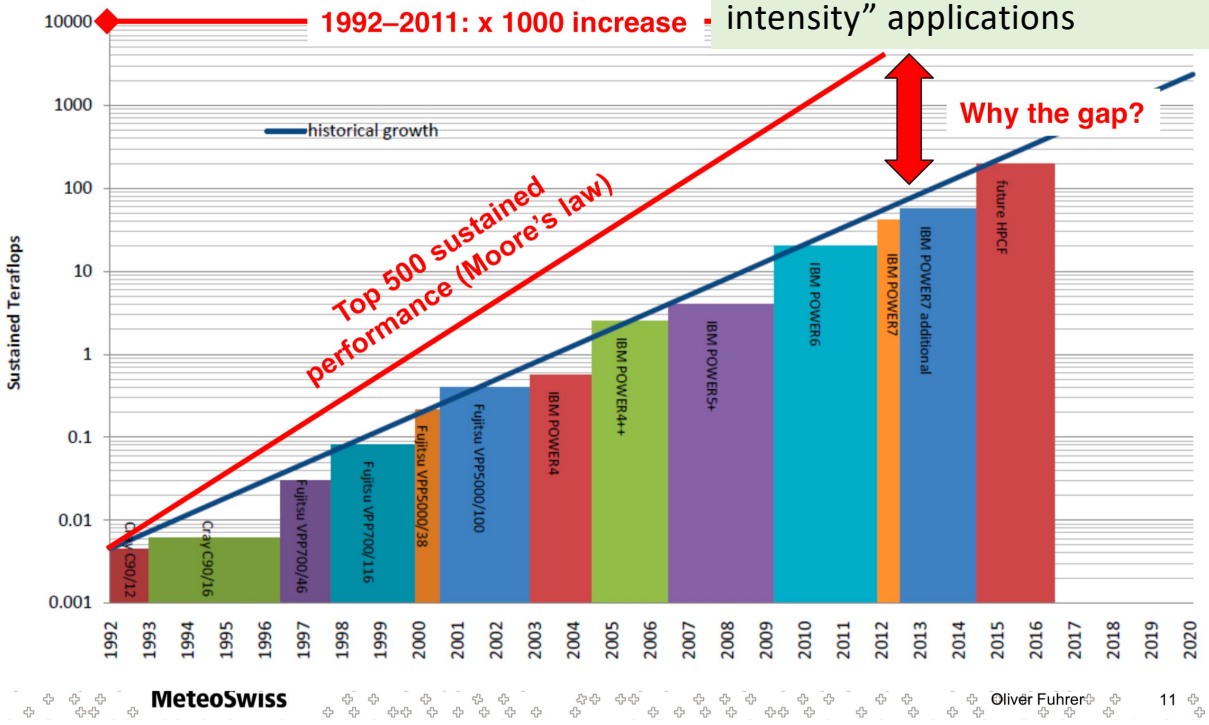
But estimates indicate that the power cost of writing data suitable for assessing the risk of extremes would be 10x larger.

**for AMIP = 22GWh (6500 households for one year) compared to 133GWh for a conventional model

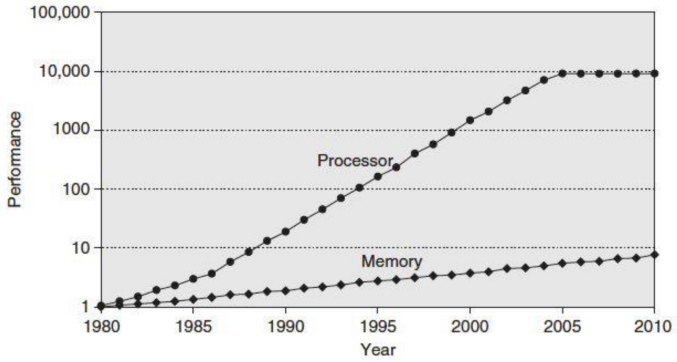
Power output of the Didcot power plant: 3'360MW

Adapting to fast-changing technology has been very challenging for our community

Thomas Schultess argues that Weather and Climate models are particular, "low-arithmetic intensity" applications

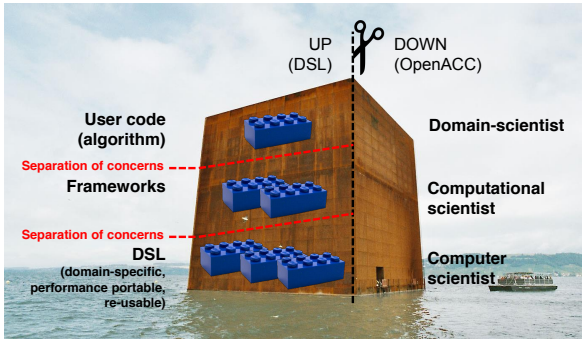


Off-chip memory bandwidth is not increasing at the same rate as FLOP performance



Trend: separation of concerns

- Example: COSMO

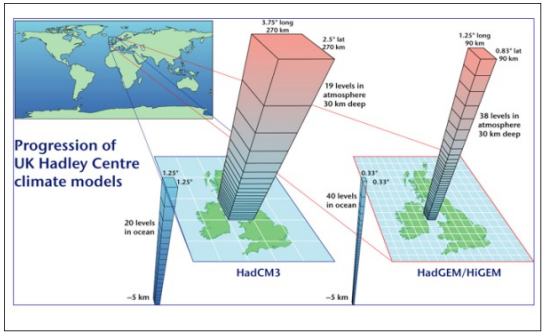


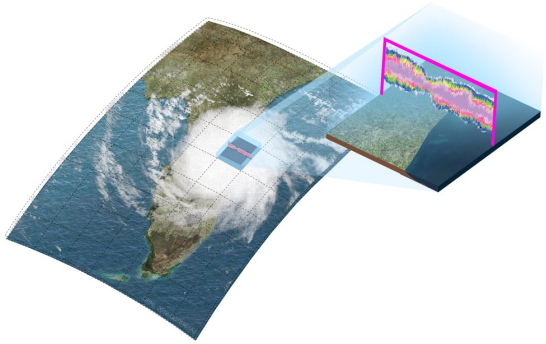
Thanks to Oliver Fuhrer, MeteoSwiss

Earth System Modelling codes and HPC

A vast number of HPC paradigms are exploited all at once:

- Shared memory parallelism
- Message-passing parallelism
- Vectorization
- Memory hierarchies
- Modularization and couplers
- Synchronous and asynchronous IO
- Use of optimised libraries (e.g. for Fourier and Legendre transforms)





And some recent trends:

1. Domain Specific Languages and frameworks (e.g. Atlas)
2. Matrix multipliers (SGEMM/DGEMM)
3. Reduced precision
4. GPUs, now going to frameworks, such as TensorFlow, Theano and JAX
5. Use of Machine Learning for emulators, data analysis, data compression, feature identification

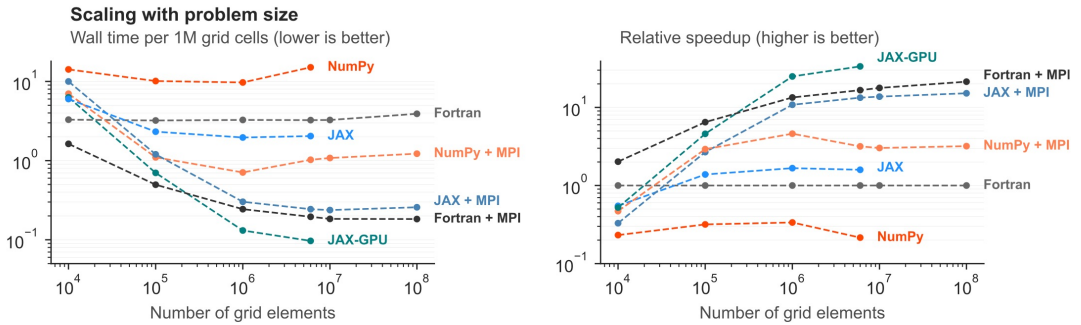


Figure 3. With JAX, Veros performance is close to native Fortran code throughout a wide range of problem sizes (number of 3D grid cells), and on both on 1 and 24 processes. Left: wall time per iteration for each backend. Right: speedup relative to single-process Fortran. Benchmarks executed on a single compute node (Table A1, architecture I).

Use of APIs to harness power of CPU and GPU

Most of us, when thinking GPU, will instantly come up with CUDA.

That is great, until one has an NVIDIA chip… and lots of $$$

There are more open ways to accelerate code than going for a vendor-specific Application Programming Interface (API).

Example from a humble Mac using JAX…

Intel XEON CPU and NVIDIA Tesla P100 in Piz Daint

Matrix multiplication using a variety of methods

Note to self on running Tensorflow with Metal

must disable conda, to use native python, and then activate the environment:

1) conda deactivate:

2) source ~/tensorflow-metal/bin/activate

Then launch Jupyter from that environment:

3) jupyter notebook

```
import numpy as np
import tensorflow as tf
devices = tf.config.list_physical_devices()
print(devices)
```

```
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'), PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
import tensorflow as tf
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

First we try to multiply with pure python, element by element

```
%%time
def explicit_matmul(a,b,c,p,q,r):
# performing the matrix multiplication
    for i in range(p):
        for j in range(r):
            curr_val = 0
            for k in range(q):
                curr_val += a[i][k]*b[k][j]
            c[i][j] = curr_val

Loops=10
size=620
a=np.array(np.random.random((size,size)), dtype=np.float32)
b=np.array(np.random.random((size,size)), dtype=np.float32)
# Just in case we change our minds about sizes
p = len(a)
q = len(a[0])
t = len(b)
r = len(b[0])
# creating the product matrix of dimensions p×r
# and filling the matrix with 0 entries
c = []
for row in range(p):
    curr_row = []
    for col in range(r):
        curr_row.append(0)
    c.append(curr_row)

for n in range(Loops):
    print("In Loop #: ", n)
    explicit_matmul(a,b,c,p,q,r)

#print(a)
#print(b)
#print(c)
```

```
In Loop #:  0
In Loop #:  1
In Loop #:  2
In Loop #:  3
In Loop #:  4
In Loop #:  5
In Loop #:  6
In Loop #:  7
In Loop #:  8
In Loop #:  9
CPU times: user 20min 9s, sys: 639 ms, total: 20min 10s
Wall time: 20min 10s
```

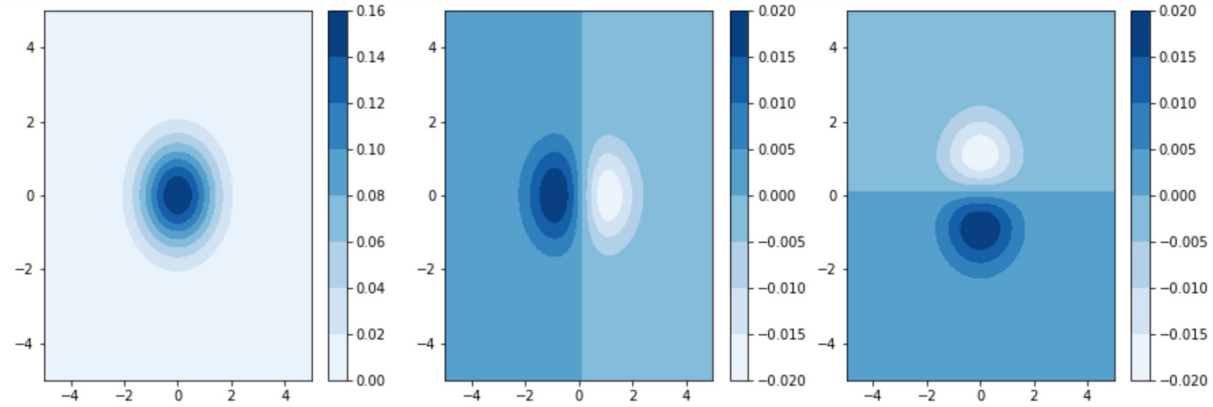
Having grown old watching the above, we try NumPy and its matmul

```
size=620
a=np.array(np.random.random((size,size)), dtype=np.float32)
b=np.array(np.random.random((size,size)), dtype=np.float32)

%timeit result=np.matmul(a,b)
#print(result)
```

```
1.01 ms ± 138 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

```
def time_matmul(x):
  #for loop in range(Loops):
    tf.matmul(x, x)
    #print(Loops," loops: {:0.2f}ms".format(1000*result/Loops))

size=620
# Force execution on CPU
print("On CPU:")
with tf.device("CPU:0"):
  x = tf.random.uniform([size, size])
  assert x.device.endswith("CPU:0")
  %timeit time_matmul(x)

# Force execution on GPU #0 if available
if tf.config.list_physical_devices("GPU"):
  print("On GPU:")
  with tf.device("GPU:0"): # Or GPU:1 for the 2nd GPU, GPU:2 for the 3rd etc.
    x = tf.random.uniform([size, size])
    assert x.device.endswith("GPU:0")
    %timeit time_matmul(x)
```

```
On CPU:
1.21 ms ± 7.46 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
On GPU:
137 µs ± 25 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```



Let us try some gradients

Once again, let us try the pure python way

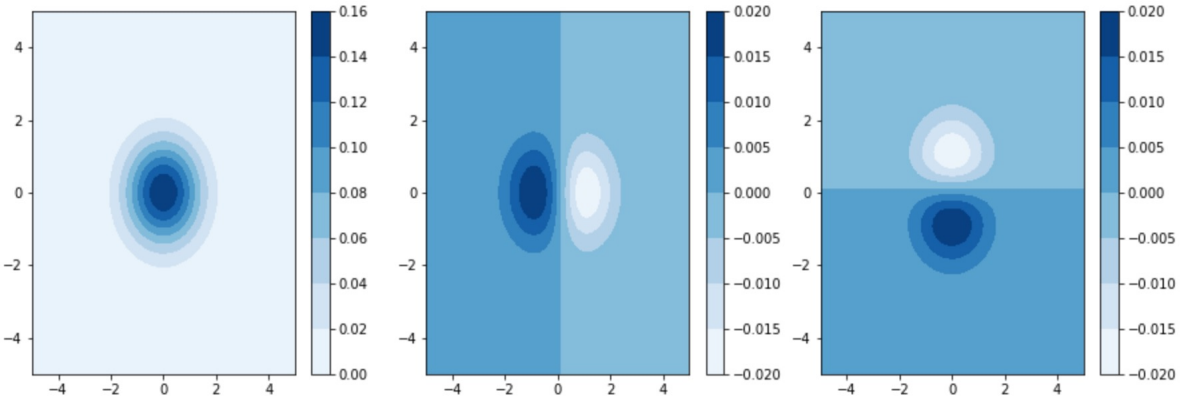
```
# define normalized 2D gaussian
def gaus2d(x=0, y=0, mx=0, my=0, sx=1, sy=1):
    return 1. / (2. * np.pi * sx * sy) * np.exp(-((x - mx)**2. / (2. * sx**2.) + (y - my)**2. / (2. *

def gaus2d_grad(z,nx,ny):
    gauss_grad_expl=np.zeros((nx,ny,2))
    for i in range(1,nx-1):
        for j in range(1,ny-1):
            gauss_grad_expl[i,j,1]=z[i,j]-z[i-1,j]
            gauss_grad_expl[i,j,0]=z[i,j]-z[i,j-1]
    return gauss_grad_expl

def gauss_plot(x,y,z,cmap='Blues'):
    plt.contourf(x, y, z, cmap=cmap)
    plt.colorbar()

def multiplot_from_generator(g, num_columns, figsize_for_one_row=None):
    # call 'next(g)' to get past the first 'yield'
    next(g)
    # default to 15-inch rows, with square subplots
    if figsize_for_one_row is None:
        figsize_for_one_row = (15, 15/num_columns)
    try:
        while True:
            # call plt.figure once per row
            plt.figure(figsize=figsize_for_one_row)
            for col in range(num_columns):
                ax = plt.subplot(1, num_columns, col+1)
                next(g)
    except StopIteration:
        pass

def plot_and_yield(z,gauss_grad_expl):
    #xs = np.linspace(-1, 1)
    x = np.linspace(-5, 5)
    y = np.linspace(-5, 5)

    # Put a 'yield' before every subplot, including the first one:
    yield
    gauss_plot(x,y,z)
    yield
    gauss_plot(x,y,gauss_grad_expl[:,:,0])
    yield
    gauss_plot(x,y,gauss_grad_expl[:,:,1])

x = np.linspace(-5, 5)
y = np.linspace(-5, 5)
x, y = np.meshgrid(x, y) # get 2D variables instead of 1D
z = gaus2d(x, y)

nx, ny = 50, 50
%timeit gauss_grad_expl = gaus2d_grad(z,nx,ny)
multiplot_from_generator(plot_and_yield(z,gauss_grad_expl), 3)
```

```
2.19 ms ± 103 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Still python, but operating on NumPy arrays, instead of single elements one by one

```
import numpy as np
from matplotlib import pyplot as plt

# define normalized 2D gaussian
def gaus2d(x=0, y=0, mx=0, my=0, sx=1, sy=1):
    return 1. / (2. * np.pi * sx * sy) * np.exp(-((x - mx)**2. / (2. * sx**2.) + (y - my)**2. / (2. * sy**2.)))

def gaus2d_grad_arr(z,nx,ny):
    gauss_grad_expl=np.zeros((nx,ny,2))
    gauss_grad_expl[:,:,1]=z-np.roll(z,1,0)
    gauss_grad_expl[:,:,0]=z-np.roll(z,1,1)
    return gauss_grad_expl

x = np.linspace(-5, 5)
y = np.linspace(-5, 5)
x, y = np.meshgrid(x, y) # get 2D variables instead of 1D
z = gaus2d(x, y)

nx, ny = 50, 50
%timeit gauss_grad_expl = gaus2d_grad_arr(z,nx,ny)
multiplot_from_generator(plot_and_yield(z,gauss_grad_expl), 3)
```

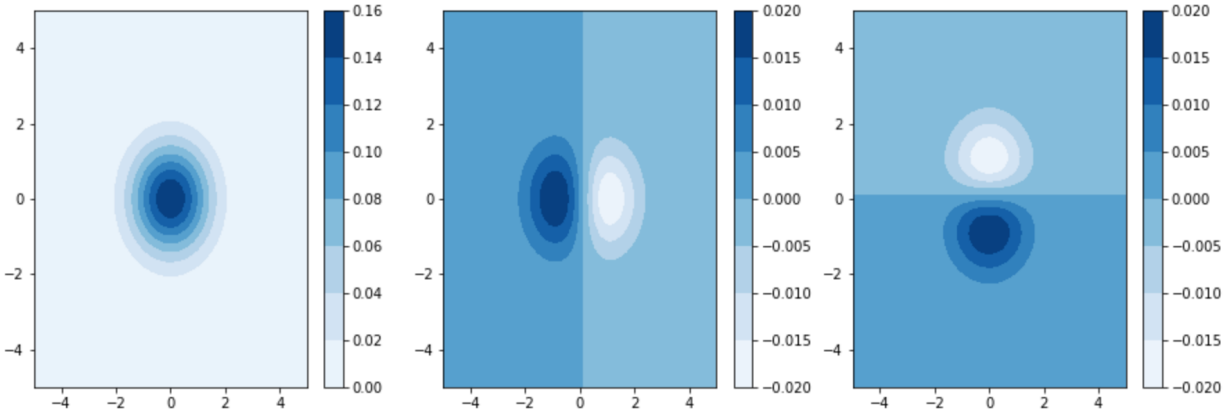
```
42.5 µs ± 248 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```



Now the NumPy way, using np.gradient

```
import numpy as np
from matplotlib import pyplot as plt

# define normalized 2D gaussian
def gaus2d(x=0, y=0, mx=0, my=0, sx=1, sy=1):
    return 1. / (2. * np.pi * sx * sy) * np.exp(-((x - mx)**2. / (2. * sx**2.) + (y - my)**2. / (2. * sy**2.)))

x = np.linspace(-5, 5)
y = np.linspace(-5, 5)
x, y = np.meshgrid(x, y) # get 2D variables instead of 1D
z = gaus2d(x, y)

%timeit gauss_grad = np.gradient(z) # calculating array of gaussian gradient
multiplot_from_generator(plot_and_yield(z,gauss_grad_expl), 3)

nx, ny = 50, 50
x1 = range(nx)
y1 = range(ny)

U = gauss_grad[1] # x component of gaussian gradient
V = gauss_grad[0] # y component of gaussian gradient
#print(" Value of U: ", U)
#print(" Value of V: ", V)

X1, Y1 = np.meshgrid(x1, y1)

fig, hf = plt.subplots(figsize=(10,10)) # setting up quiver plot
hf.set_title("Example - gradient of gaussian- every 2nd arrow")
Q = hf.quiver(X1[::2, ::2], Y1[::2, ::2], U[::2, ::2], V[::2, ::2], unit
    pivot='mid', scale=0.1, headwidth=5)
qk = hf.quiverkey(Q, 0.9, 0.9, 0.1, r'$2d \ vector \ plot$', labelpos='E
    coordinates='figure')
hf.scatter(X1[::2, ::2], Y1[::2, ::2], color='c', s=0)
plt.show()
```

```
50 µs ± 915 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```



Now NumPy on steroids: the JAX way imitating NumPy, and it can do it on the GPU (currently JAX is disabled until I rebuild with Metal+Tensorflow). Should work fine on CUDA GPU and a standard JAX installation.

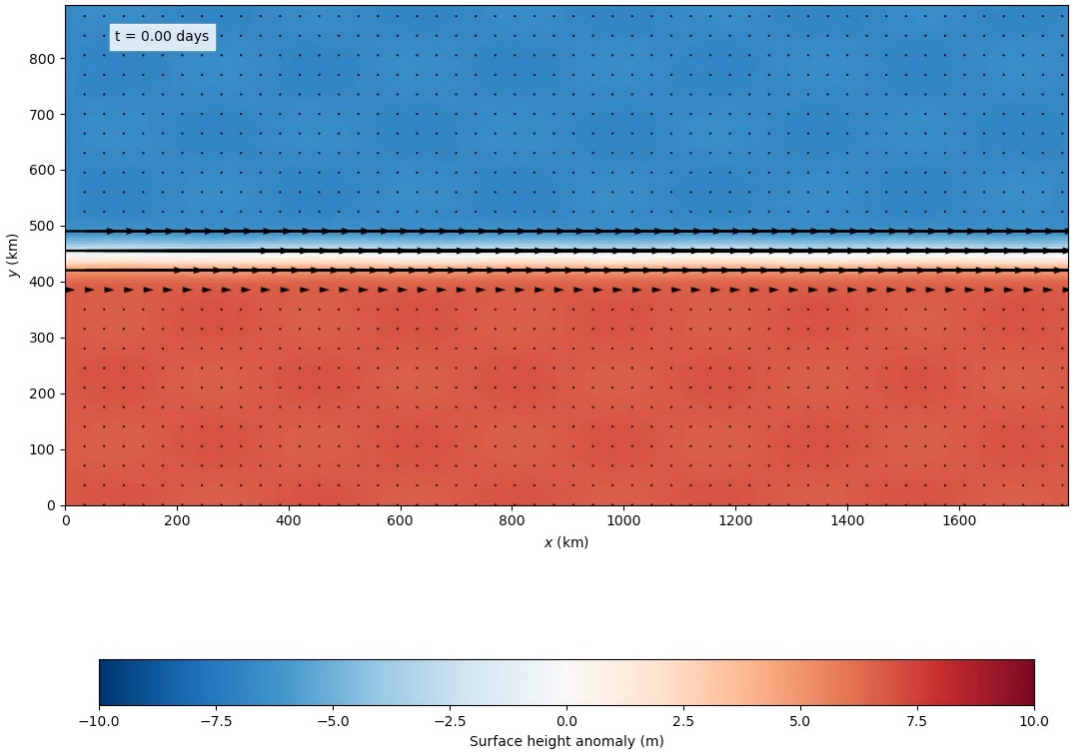
```
import numpy as np
#import jax.numpy as np
#import jax.numpy as jnp
from matplotlib import pyplot as plt

# define normalized 2D gaussian
def gaus2d(x=0, y=0, mx=0, my=0, sx=1, sy=1):
    return 1. / (2. * np.pi * sx * sy) * np.exp(-((x - mx)**2. / (2. * sx**2.) + (y - my)**2. / (2. * sy**2.)))

x = np.linspace(-5, 5)
y = np.linspace(-5, 5)
x, y = np.meshgrid(x, y) # get 2D variables instead of 1D
z = gaus2d(x, y)

%timeit gauss_grad = np.gradient(z) # calculating array of gaussian gradient
```

```
40.2 µs ± 482 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

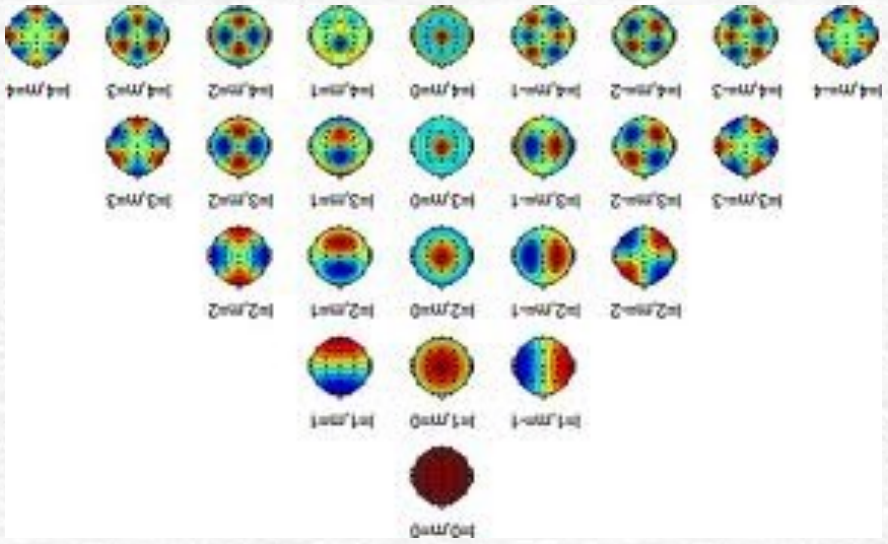
MPI examples with SWE

Running JAX-enabled Shallow Water Model with mpi

Command	MPI processes	CPU Seconds per Model Day	Elapsed	Threads
`mpirun -n 1 python shallow_water_jax.py`	1	4.33	258.87s	64
`mpirun -n 2 python shallow_water_jax.py`	2	4.31	256.94s	128
`mpirun -n 4 python shallow_water_jax.py`	4	2.5	153.22s	256
`mpirun -n 8 python shallow_water_jax.py`	8	2.17	128.53s	
`mpirun -n 16 python shallow_water_jax.py`	16	2.66	156.62s	
REFERENCE: pure python, with JAX disabled		2878.59s		



Greater Accuracy with Less Precision?

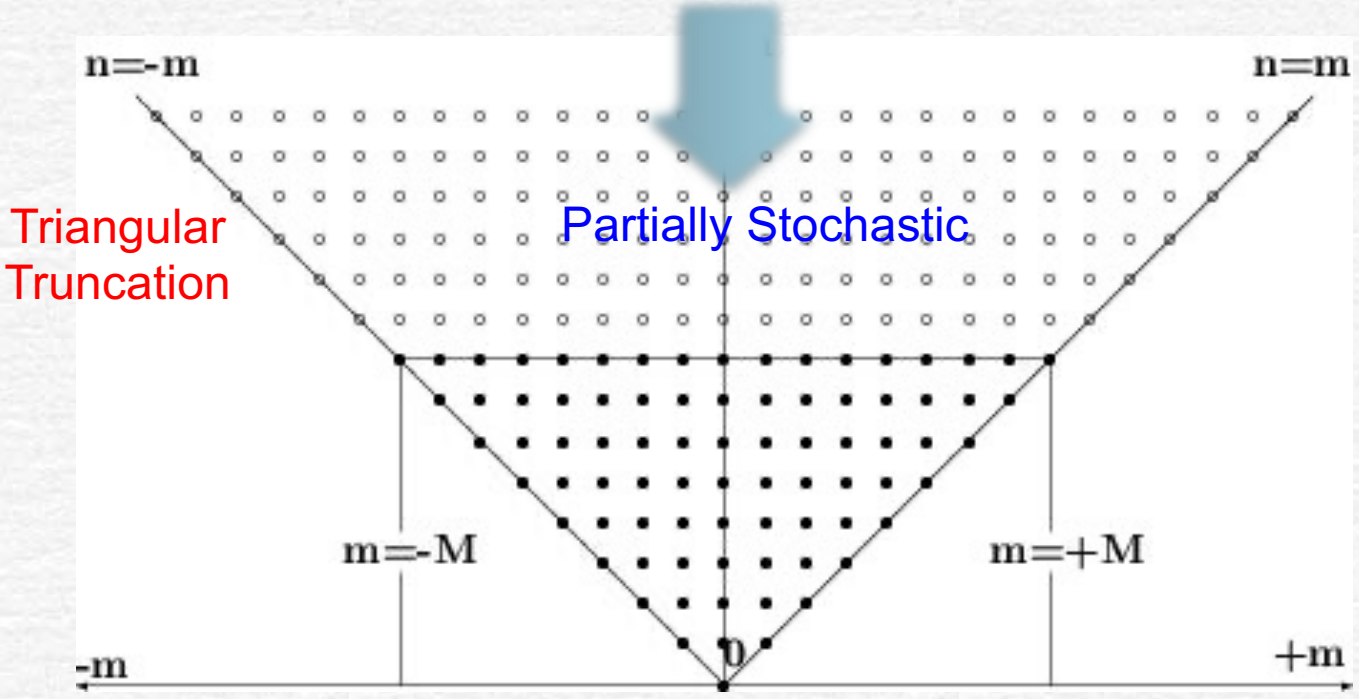


Use freed-up computing resource to extend simulator to cloud resolved scales?

Decreasing precision, and determinism

T. Palmer et al. ECMWF / Oxford

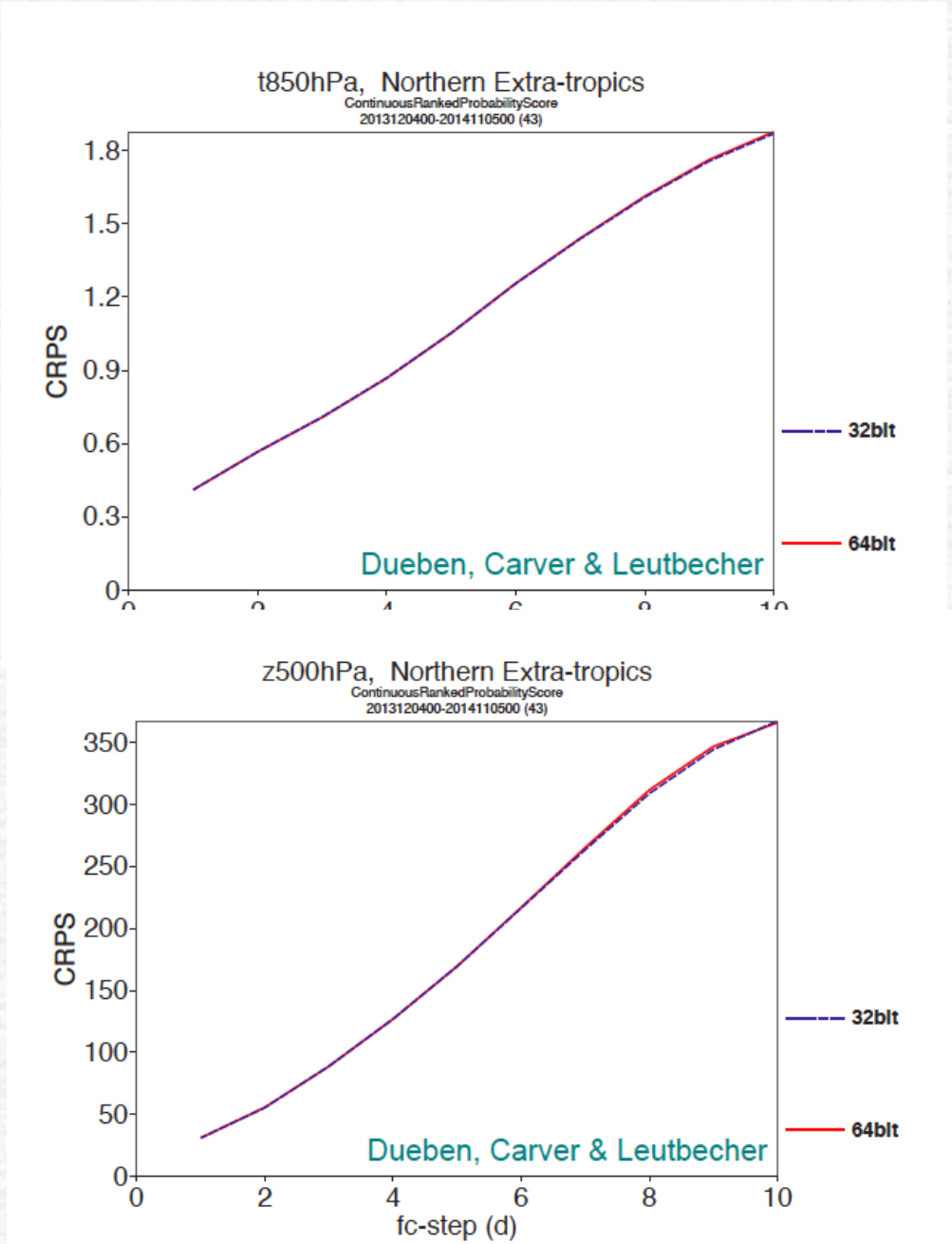
Stochastic Parametrisation



If parametrisation is partially stochastic, are we "over-engineering" our models (parametrisations, dynamical core) by using double precision bit-reproducible computations throughout?

Are we making inefficient use of computing resources that could otherwise be used to increase resolution?

T. Palmer et al. ECMWF / Oxford



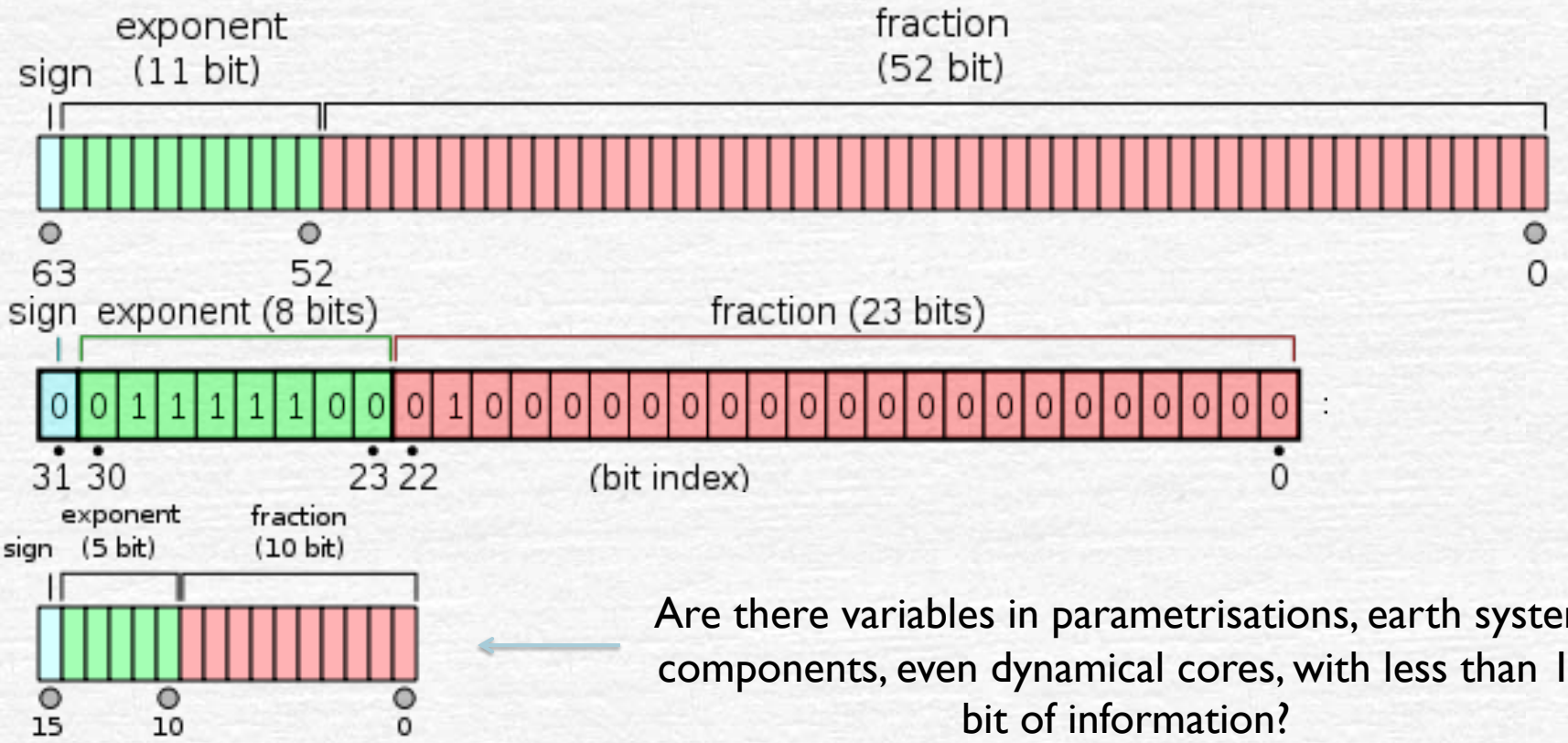
IFS: Single vs Double Precision

T399 20 member IFS

Can run 15 day T639 at single precision for cost of 10-day T639 at double precision

T. Palmer et al. ECMWF / Oxford

Reduced precision allows computations to proceed more quickly, and data to be moved with less energy overhead.



Are there variables in parametrisations, earth system components, even dynamical cores, with less than 16 bit of information?

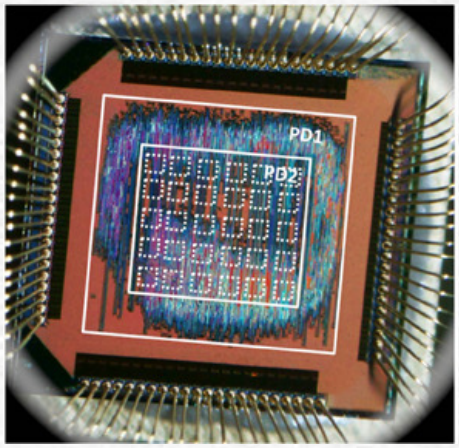
We need to lobby computer manufacturers to allow "mixed precision" – including 16 bit – computation.

T. Palmer et al. ECMWF / Oxford

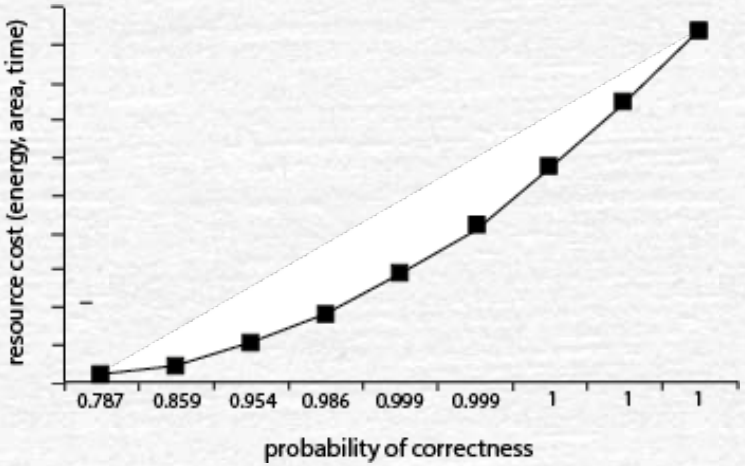
Superefficient inexact chips

http://news.rice.edu/2012/05/17/computing-experts-unveil-superefficient-inexact-chip/

Prototype
Probabilistic
CMOS
Chip



Krishna Palem.
Rice University

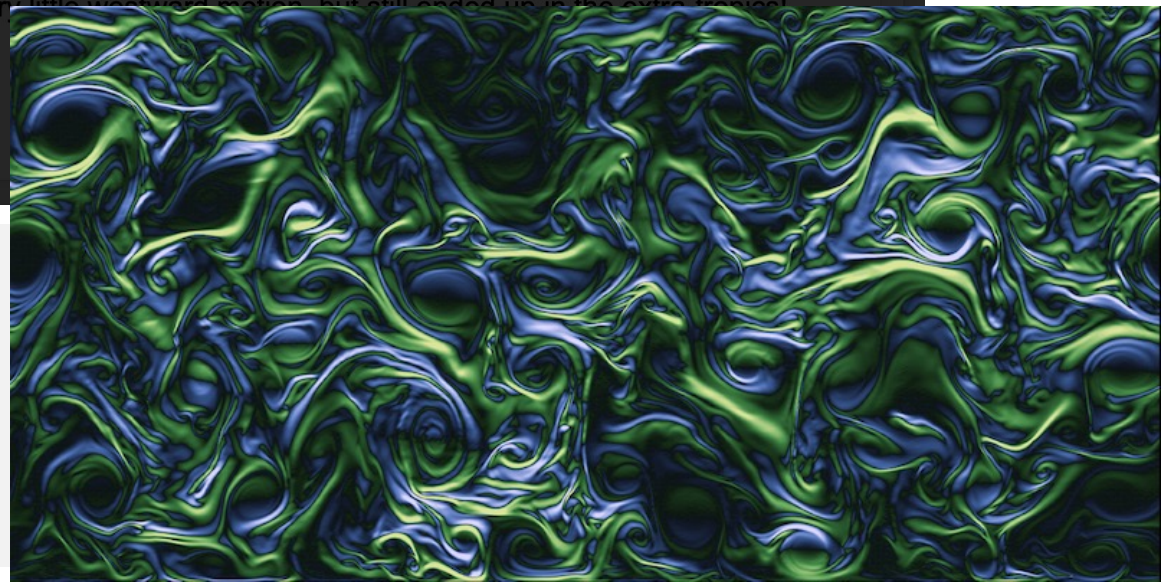


The chip that produced the frame with the most errors (right) is about 15 times more efficient in terms of speed, space and energy than the chip that produced the pristine image (left).

T. Palmer et al. ECMWF / Oxford

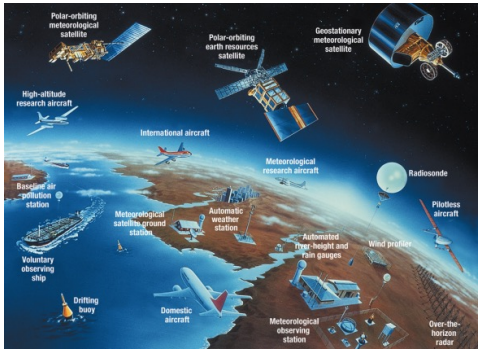
An example of reduced precision in practice (ShallowWaters, M Klüwer)

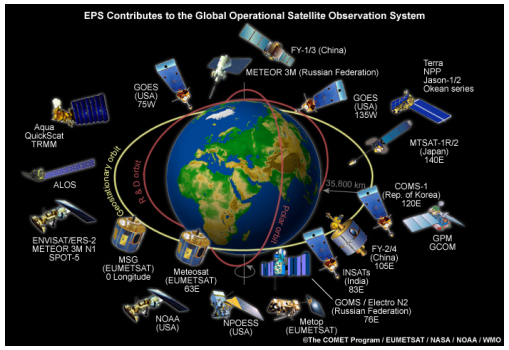
```
julia> P = run_model(Float16,Ndays=3000,nx=100,L_ratio=1,bc="nonperiodic",wind_forcing_x="double_gyre",topography="flat",output=true);
Starting ShallowWaters run 6 on Wed, 02 Mar 2022 17:06:24
Model integration will take approximately 3min, 1s,
and is hopefully done on Wed, 02 Mar 2022 17:09:25
70% NaNs detected at time step 35856
 Integration done in 2min, 15s.
All data stored.

julia> P = run_model(Float32,Ndays=3000,nx=100,L_ratio=1,bc="nonperiodic",wind_forcing_x="double_gyre",topography="flat",output=true);
Starting ShallowWaters run 7 on Wed, 02 Mar 2022 17:09:09
Model integration will take approximately 20.0s,
and is hopefully done on Wed, 02 Mar 2022 17:09:29
100% Integration done in 30.4s.
All data stored.

julia> P = run_model(Float64,Ndays=3000,nx=100,L_ratio=1,bc="nonperiodic",wind_forcing_x="double_gyre",topography="flat",output=true);
Starting ShallowWaters run 8 on Wed, 02 Mar 2022 17:10:01
Model integration will take approximately 42.0s,
and is hopefully done on Wed, 02 Mar 2022 17:10:43
100% Integration done in 52.7s.
All data stored.

julia>
```

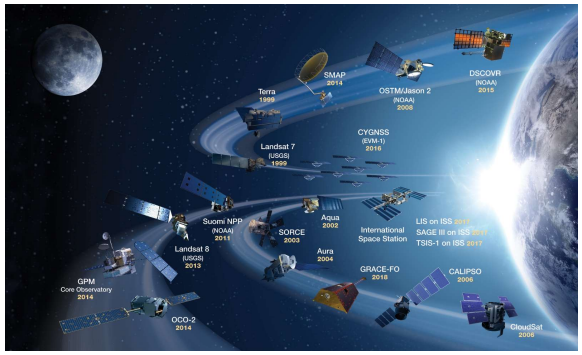


Note that the 16-bit is emulated, so there is a huge overhead on my Mac. The real comparison is going from 32 to 64 bit, with a cost increase of about 72%. Results indistinguishable.

New computers will support 16 and 8 bit in hardware. The energy costs are far less than for traditional computers.







Abstraction of current NWP workflow

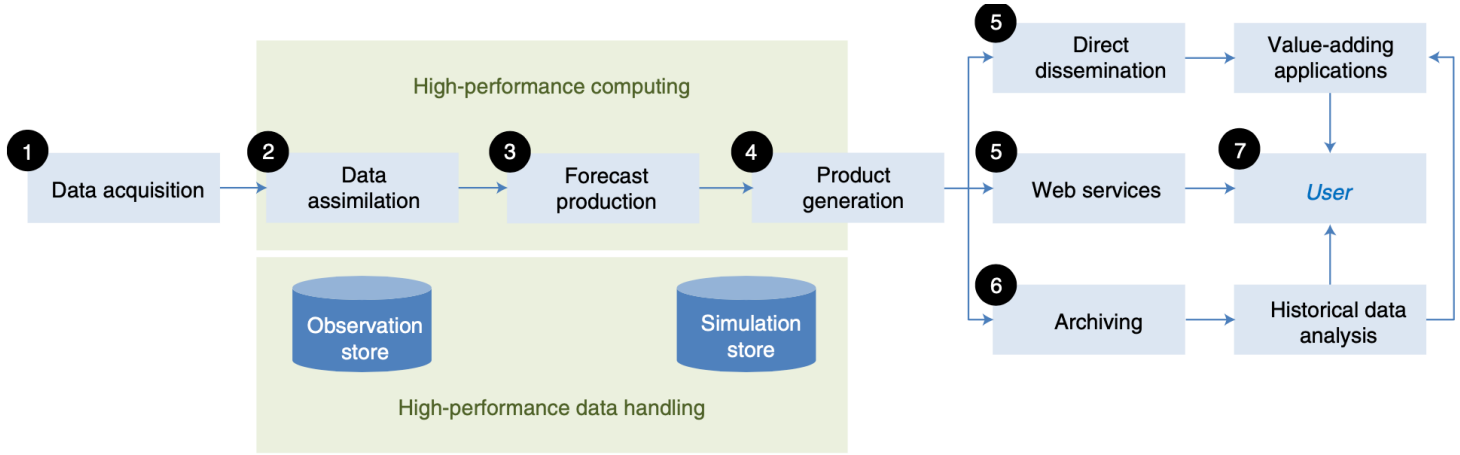
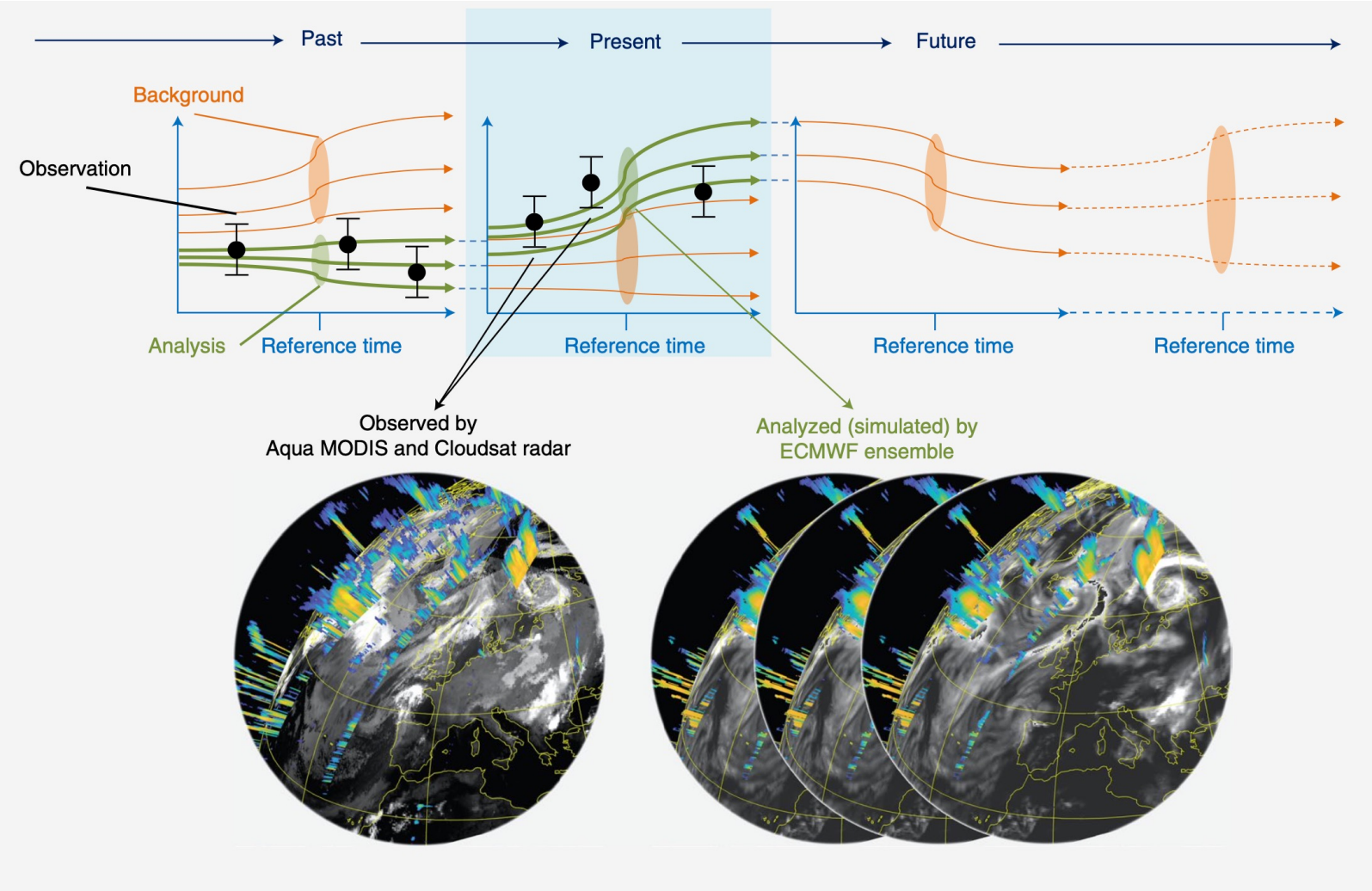


Fig. 1 | Typical production workflow in operational numerical weather prediction. (1) High-volume and high-speed observational data acquisition and pre-processing; (2) data assimilation into models to produce initial conditions for forecasts; (3) forecast production by Earth-system simulation models; (4) generation of output products tailored to the portfolio of weather and climate information users; (5) direct dissemination of raw output and web-products; (6) long-term archiving for reuse in statistical analyses and performance diagnostics; (7) user-specific applications and data-driven analytics.

The challenge is far worse for Digital Twins



The Earth-system digital twin, shown in the figure, optimally combines simulations and near-real-time observations to monitor the evolution of the Earth system.

Massively increased IO requirements alongside resolution, DA, complexity…

Digital Twins for weather and climate: workflow

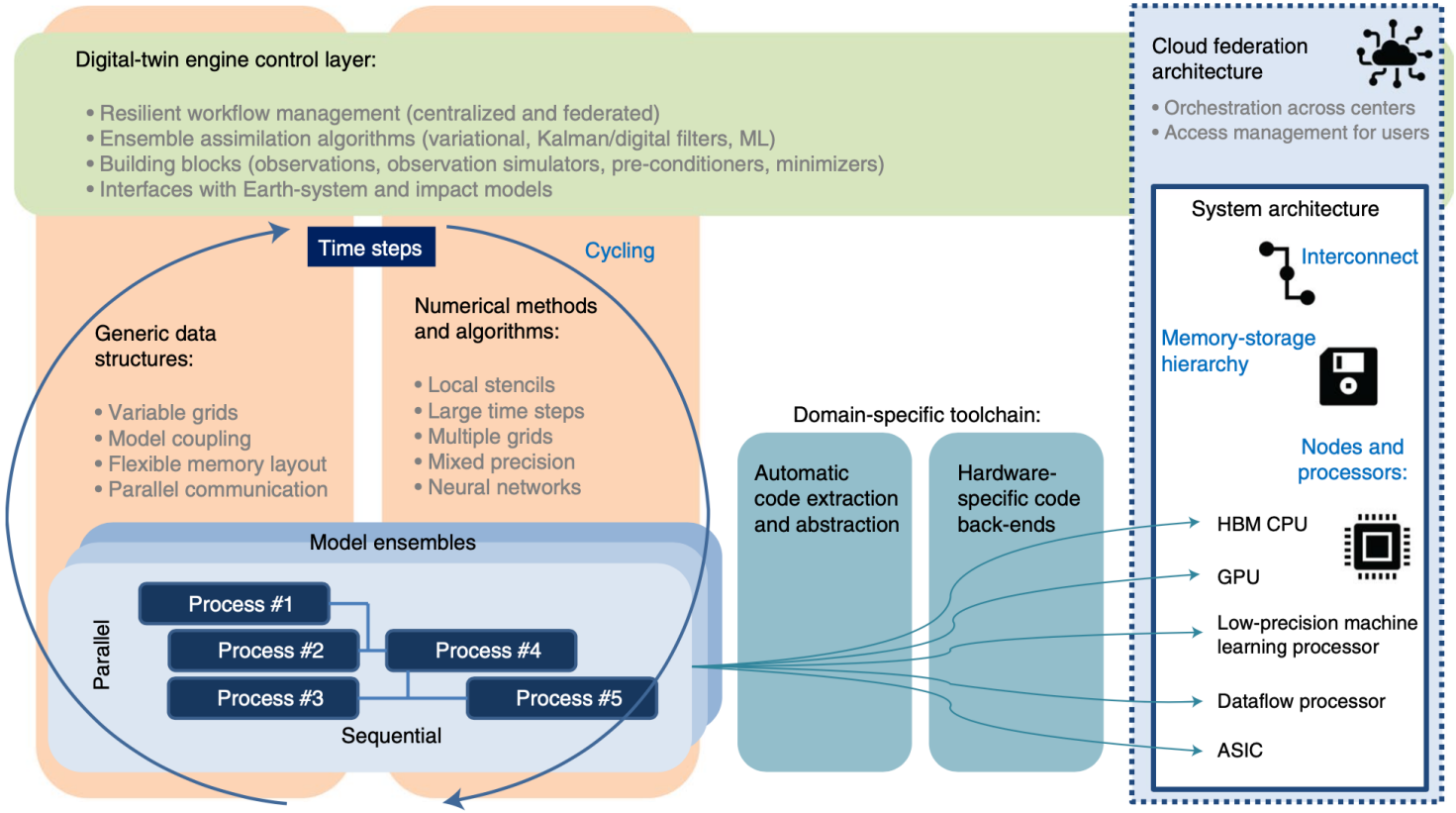


Fig. 3 | Conceptual view of an efficient software infrastructure for the Earth-system digital twin. The digital-twin control layer drives flexible workflows for Earth-system modeling and data assimilation using generic data structures and physical process simulations that exploit parallelism and are based on algorithms minimizing data movement. DSLs map the algorithmic patterns optimally on the memory and parallel processing power of heterogeneous processor architectures. The computing architecture is based on heterogeneous, large-scale architectures within federated systems.

Bauer et al. Nature Comp 2021

The race to exaxcale computing and DTS

- Weather and Climate models are a good example of applications that aim to exploit exascale.

Reflecting on the Goal and Baseline for Exascale Computing: A Roadmap Based on Weather and Climate Simulations

Thomas C. Schulthess
ETH Zurich, Swiss National Supercomputing Centre
Peter Bauer
European Centre for Medium-Range Weather Forecasts
Nils Wedi
European Centre for Medium-Range Weather Forecasts
Oliver Fuhrer
MeteoSwiss
Torsten Hoefler
ETH Zurich
Christoph Schär
ETH Zurich

- Schultess et al. argue that "This analysis is also representative of other simulation domains that rely on grid-based solvers of partial differential equations, such as seismology and geophysics, combustion and fluid dynamics in general, solid mechanics, materials science, or quantum chromodynamics. "

- They are "low arithmetic intensity" applications.
- The roadmap is:
  - Long-term scientific aim: a 100m global model
  - The practical interim goal post is a 1km global model
    - **This would require a ~100-fold improvement in speed**
    - Thomas argues that it is possible to combine architecture design and algorithmic design to come credibly close to a factor of ~4
      - More homogeneous grid: 4x
      - New generation GPUs with better memory bandwidth: 2x
      - Better scaling: 3x

Table 1. Ambitious target configuration for global weather and climate simulations with km-scale horizontal resolution accounting for physical Earth-system processes, and with today's computational throughput rate.

Horizontal resolution	1 km (globally quasi-uniform)
Vertical resolution	180 levels (surface to ~100 km)
Time resolution	0.5 min
Coupled	Land-surface/ocean/ocean-waves/sea-ice
Atmosphere	Non-hydrostatic
Precision	Single or mixed precision
Compute rate	1 SYPD (simulated years per wall-clock day)

We are ~100x short of the required speed
How to get there with new architectures

Table 1 | Stepwise estimate of hybrid CPU–GPU machine size for digital-twin computing based on COSMO benchmark[18,30] accounting for known, near-future technology upgrades and methodological redesign as described in this paper

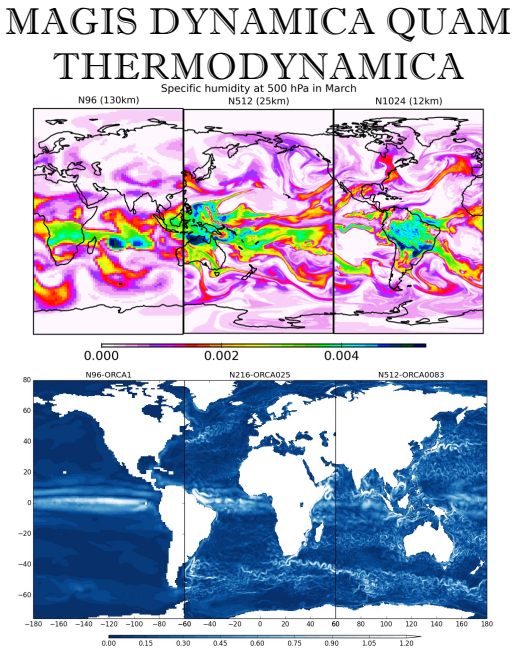
Upgrade and technology	Acceleration factor	Remaining shortfall factor	Reference
5,000 Intel Xeon E5-2690 v3/Nvidia P100	1	100	[18]
Data structures, grids, numerical methods, mixed precision, machine learning	4	25	[18]
GPU bandwidth efficiency and bandwidth	2	13	Example, NVIDIA V100 vs P100[104]
GPU peak bandwidth and memory	1.5	8	Example, NVIDIA A100 vs V100[105]
High-bandwidth memory	2	4	Example, HBM3 vs HBM2[106]

Acceleration and shortfall factors describe the expected acceleration of a digital-twin benchmark delivered by each technology upgrade and the remaining shortfall of achievable time to solution, respectively.

Bauer et al. Nature Comp 2021

Summary

- The current goals of the physical climate modelling community require an international, coordinated approach, particularly so for analysis and scientific publication. ESIWACE is an example. **We are a factor 100 short of the required speed, not to talk about efficiency.**
  - We could continue to work in distributed mode, at national TIER-1 or TIER-0 centres, as long as we adopt strict experimental protocols. Joint analysis, however, will suffer, and so will scientific output.
  - Joining up the compute and the analysis phases of each experiment would create an economy of scale
  - Working together at a larger, dedicated centre will give us far better control of machine configurations and management (e.g. the queues).
- Digital Twins are both a formidable challenge and an opportunity, especially to co-design with computational scientists and with users, and share results (one way to go green).
- The machines we need do not yet exist:
  - Separation of concerns (e.g. DSLs) and a combination of other strategies (new architectures, algorithms, etc.) can help.
- We must reduce our power footprint very substantially: methods, but also experimental designs
- We need to educate and provide incentives to a new class of scientists that can better bridge the science and technology expertise required to tackle these daunting challenges



Emerging processes in the atmosphere and ocean as model resolution is increased

University of Reading

p.l.vidale@reading.ac.uk

https://hrcm.ceda.ac.uk

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL