efficient computing

at the

netherlands eScience center

Dr. Jason Maassen

October 2nd 2017

My background

Jason Maassen

Computer Science Dept. of VU University Amsterdam
…. in Henri's group!:

1993-1998 MSc: Fast Parallel Java
1998-2003 PhD: Method Invocation Based Prog. Models for Parallel Prog. in Java
2002-2012 Postdoc: Ibis + (GridLab, StarPlane, VL-e, PROMM-Grid, …)

Netherlands eScience Center:

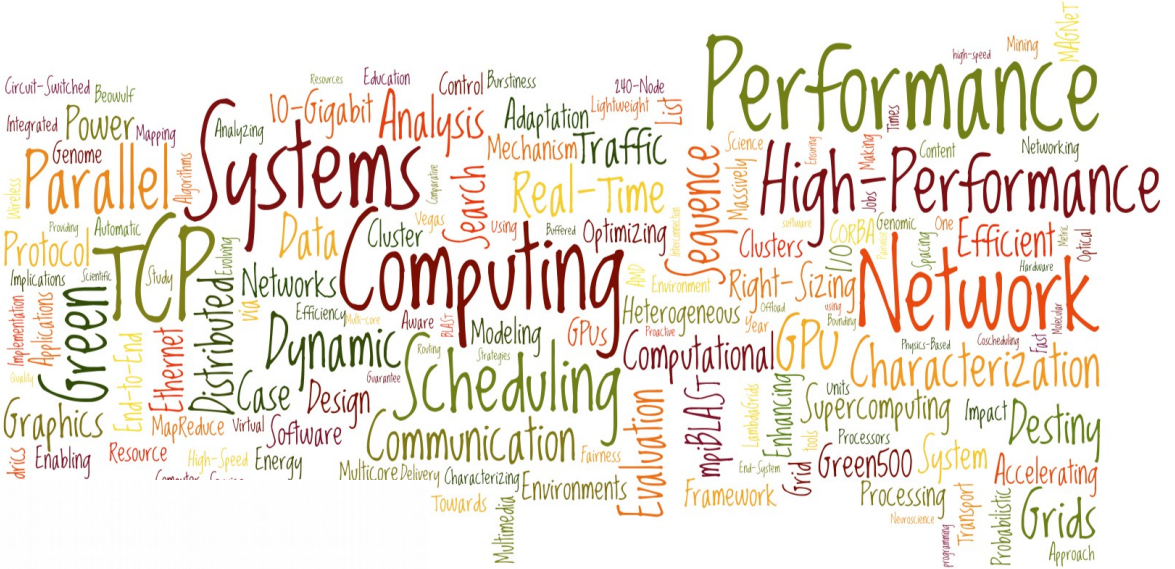
2012-2013: eScience Research Engineer
2013-2016: eScience Coordinator
2016-now: Technology Lead – Efficient Computing

netherlands eScience center ?

Research institute founded by NWO and SURF in 2011

“Enabling digitally enhanced research through efficient utilization of data, software and e-infrastructure”

Science is changing



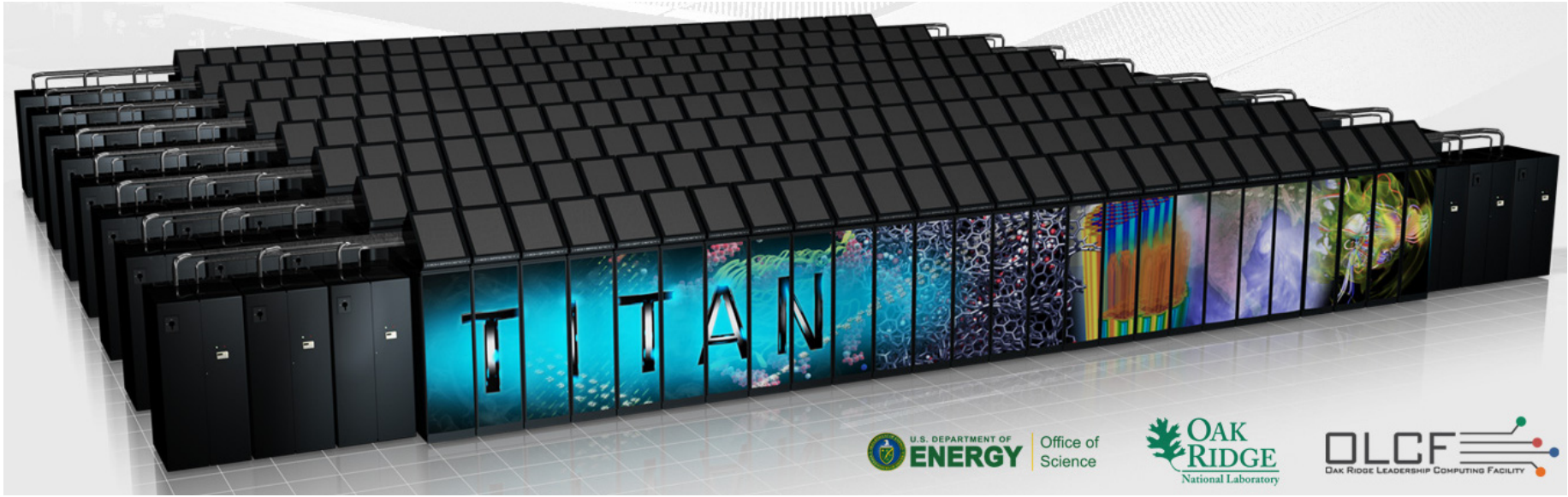
“Big data” in the real world

For **a few** scientists “big data” is a 20 PB storage cluster
(like this JUST cluster at Julich Supercomputing Center)

“Big data” in the real world

For **most** scientists “big data” is simply more than fits on their USB stick

“Big compute” in the real world



A **few** scientists run their computations on big machines.
(like this 560640 core,17 PFlop/s Titan at Oak Ridge National Laboratory)

“Big compute” in the real world

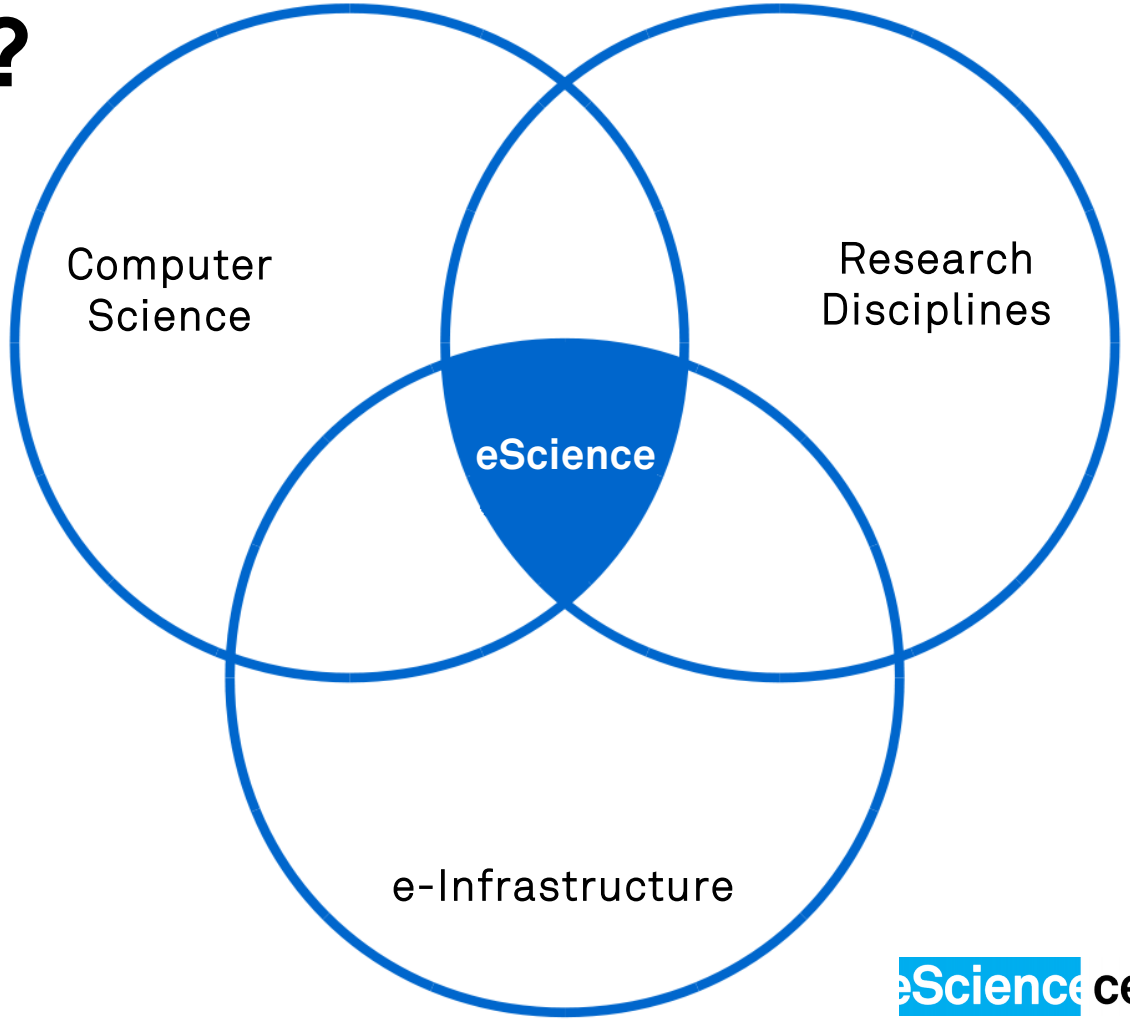
Most scientists simply stick to whatever their laptop can compute over the weekend.

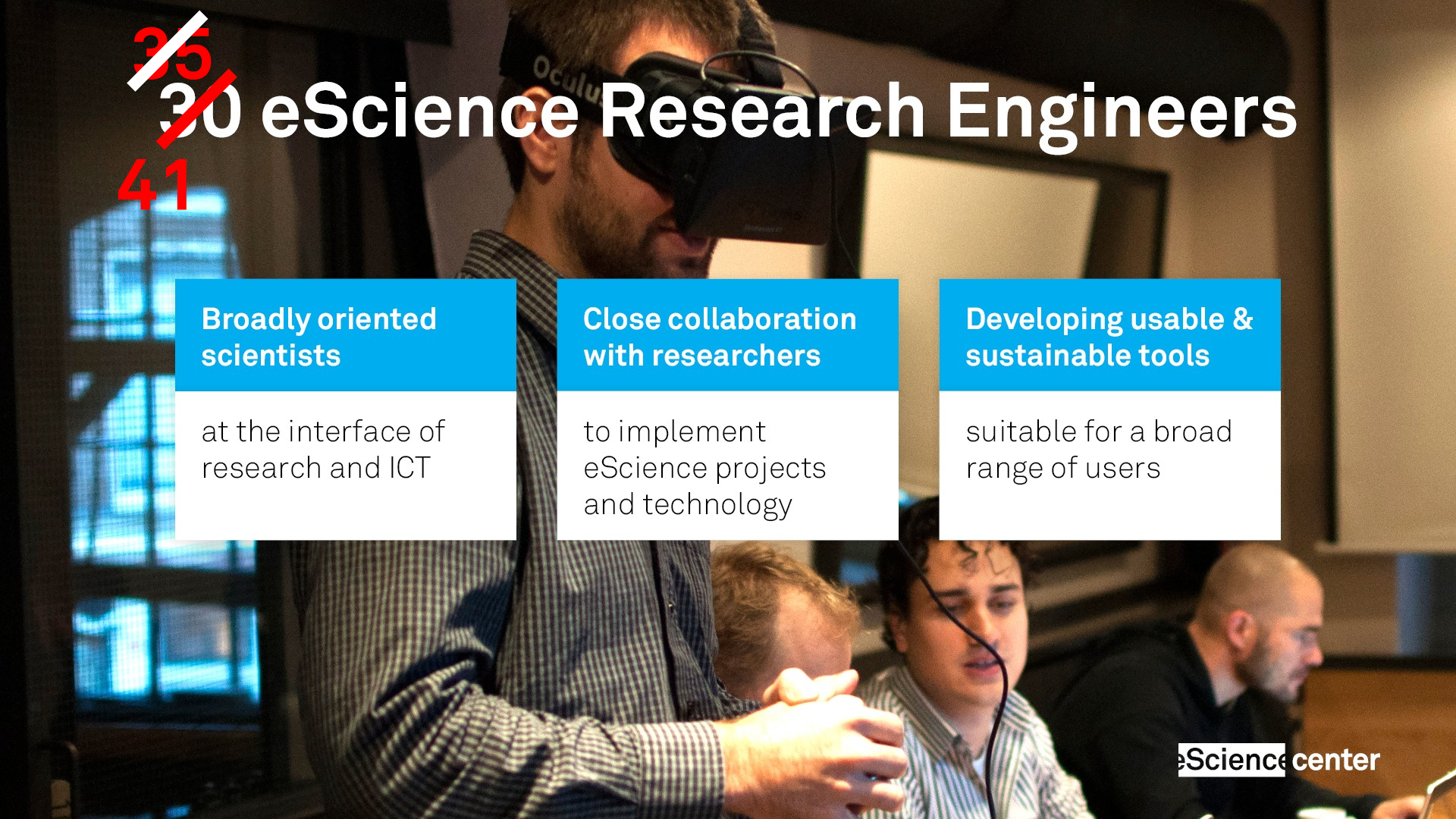
Why eScience

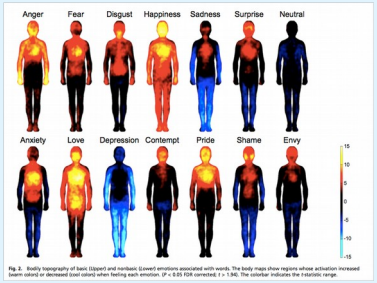
Bridging the gap between:
science and infrastructure
science and computer science
scientific domains

eScience center

eScience?


Computer Science
Research Disciplines
eScience
e-Infrastructure


35
30
41
30 eScience Research Engineers
Broadly oriented scientists
at the interface of research and ICT
Close collaboration with researchers
to implement eScience projects and technology
Developing usable & sustainable tools
suitable for a broad range of users
eScience center

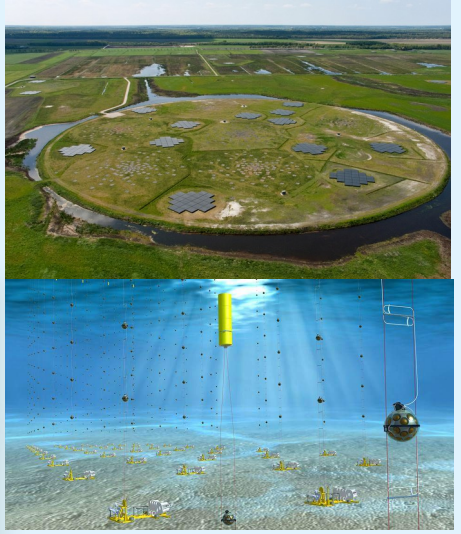


so far: 94 projects

(on many different topics)

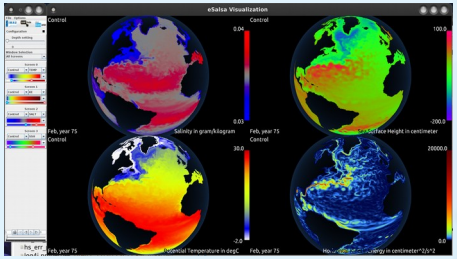
Humanities & Social Sciences

incl. SMART cities, text analysis, creative technologies



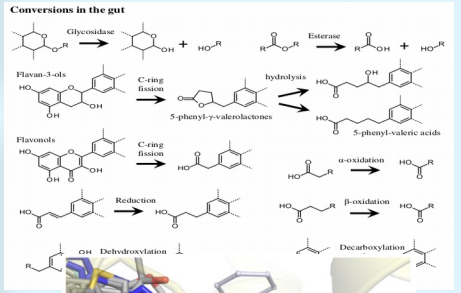
Physics & Beyond

incl. astronomy, high-energy physics, advanced materials



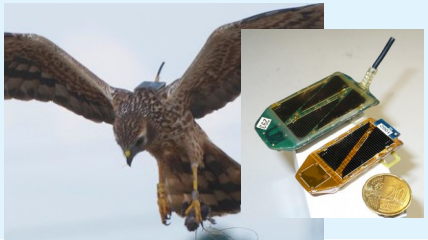
Sustainability & Environment

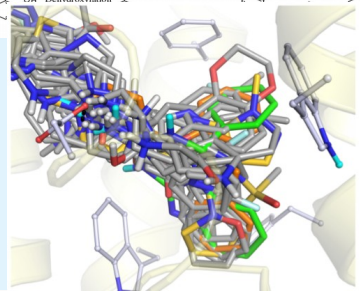
incl. climate, ecology, energy, logistics, water management



Life Sciences & eHealth

incl. bio-imaging, next generation sequencing, molecules





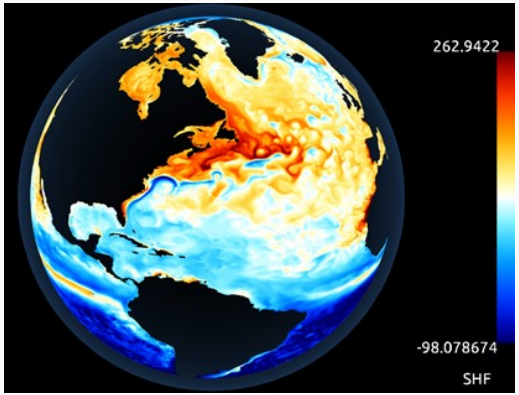
efficient computing ?

sheer number of computations

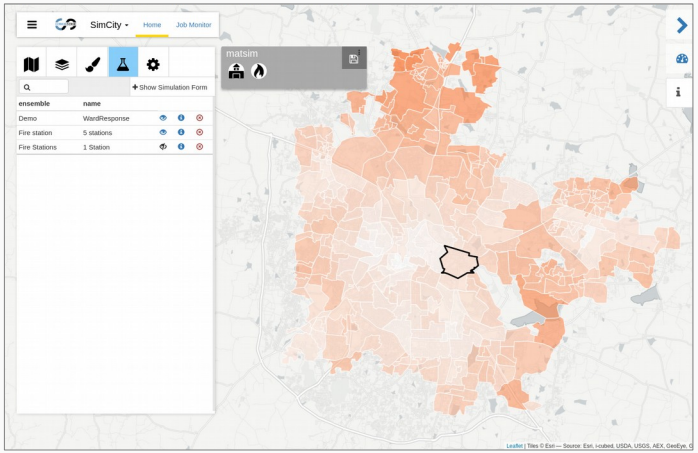
speed & volume of incoming data

efficient computing ?

ease of use



efficient computing ?



high performance computing

real time streaming sensor data

efficient computing ?

deployment and coupling

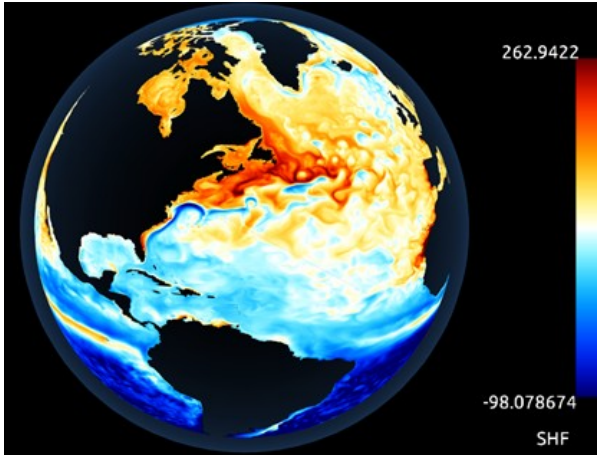
A selection of

projects

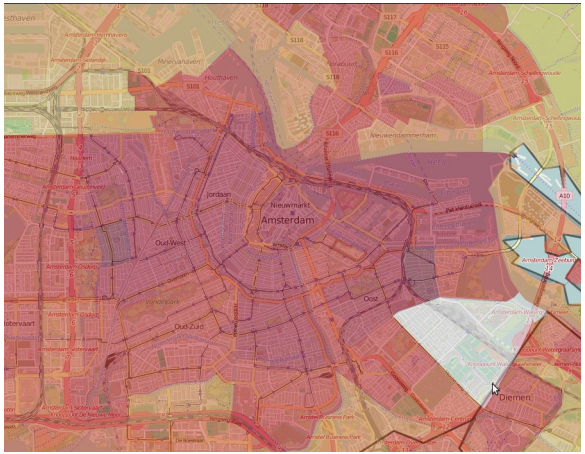
using

high performance (distributed) computing

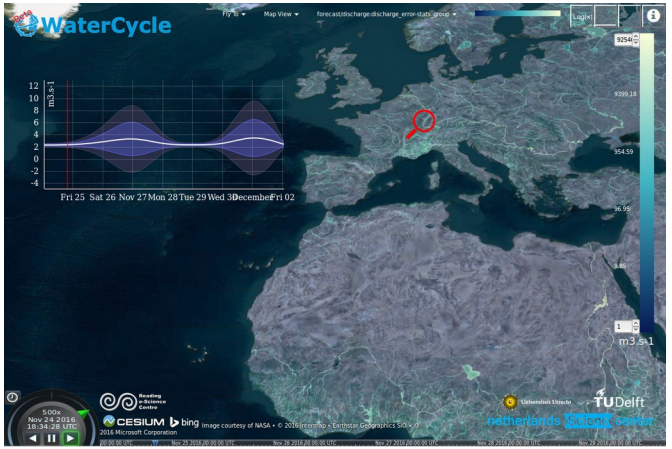
The quest for resolution



eSalsa



Summer in the city



eWaterCycle

Many of our “traditional HPC” projects have a climate focus. They need to increase the resolution of their simulations, couple models, integrate observation data, after which they have trouble with load balancing or the large amounts of data they need to store

netherlands eScience center

Potential solutions

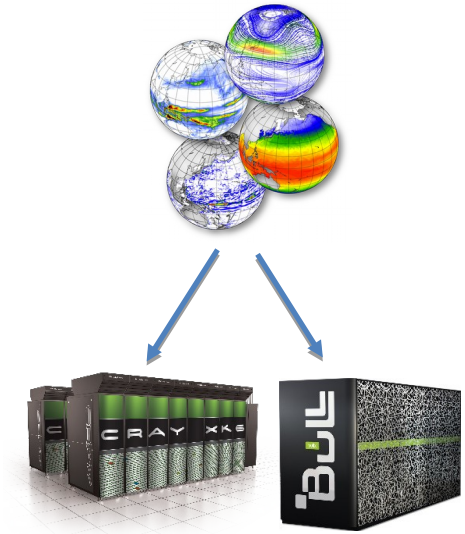
```
218
219 !$OMP PARALLEL DO SCHEDULE(STATIC)
220 DO K=1,KBAR
221    DO J=1,JBAR
222       DO I=1,IBAR
223          IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
224          FRHO(I,J,K) = -FRHO(I,J,K)
225                      + (FX(I,J,K,0)*UU(I,J,K)*R(I)-FX(I-1,J,K,0)*UU(I-1,J,K)*
226                      + (FY(I,J,K,0)*VV(I,J,K)      -FY(I,J-1,K,0)*VV(I,J-1,K)
227                      + (FZ(I,J,K,0)*WW(I,J,K)      -FZ(I,J,K-1,0)*WW(I,J,K-1)
228       ENDDO
229    ENDDO
230 ENDDO
231 !$OMP END PARALLEL DO
232
233 SPECIES LOOP: DO N=1,N_TRACKED_SPECIES
```

Parallel Begins

Parallel Ends

parallelization

GPUs



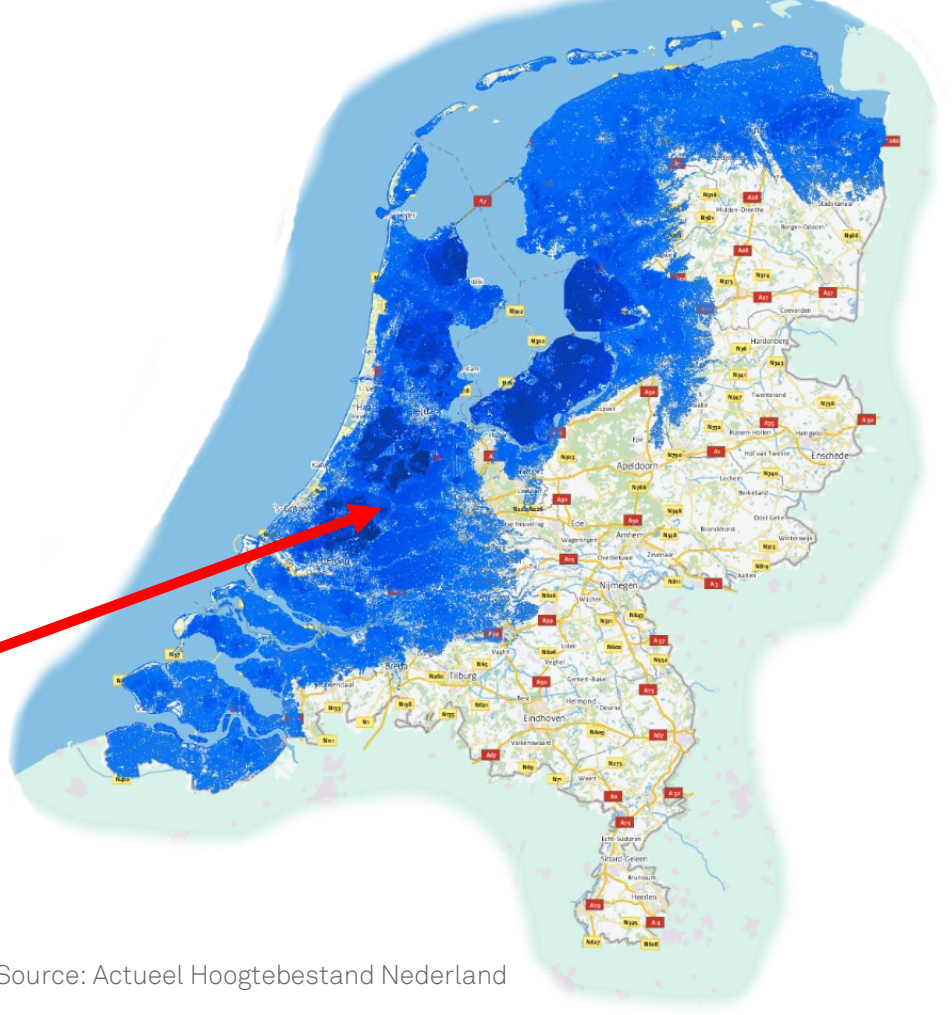
distributed computing

GPUs deliver more performance per node than parallelization, but require a significant effort to rewrite existing code. Distributed work over multiple machines is easier, but results in less performance improvement and requires additional tools.

The eSalsa Project

Gain insight into **regional** sea-level changes (caused by climate change) by simulating the oceans with an unprecedented level of detail.

26% to 55% below sea level



Source: Actueel Hoogtebestand Nederland

IMAU Institute for Marine and Atmospheric research Utrecht

Universiteit Utrecht

VU UNIVERSITY AMSTERDAM

COMMIT/

netherlands eScience center

Sea levels are changing…

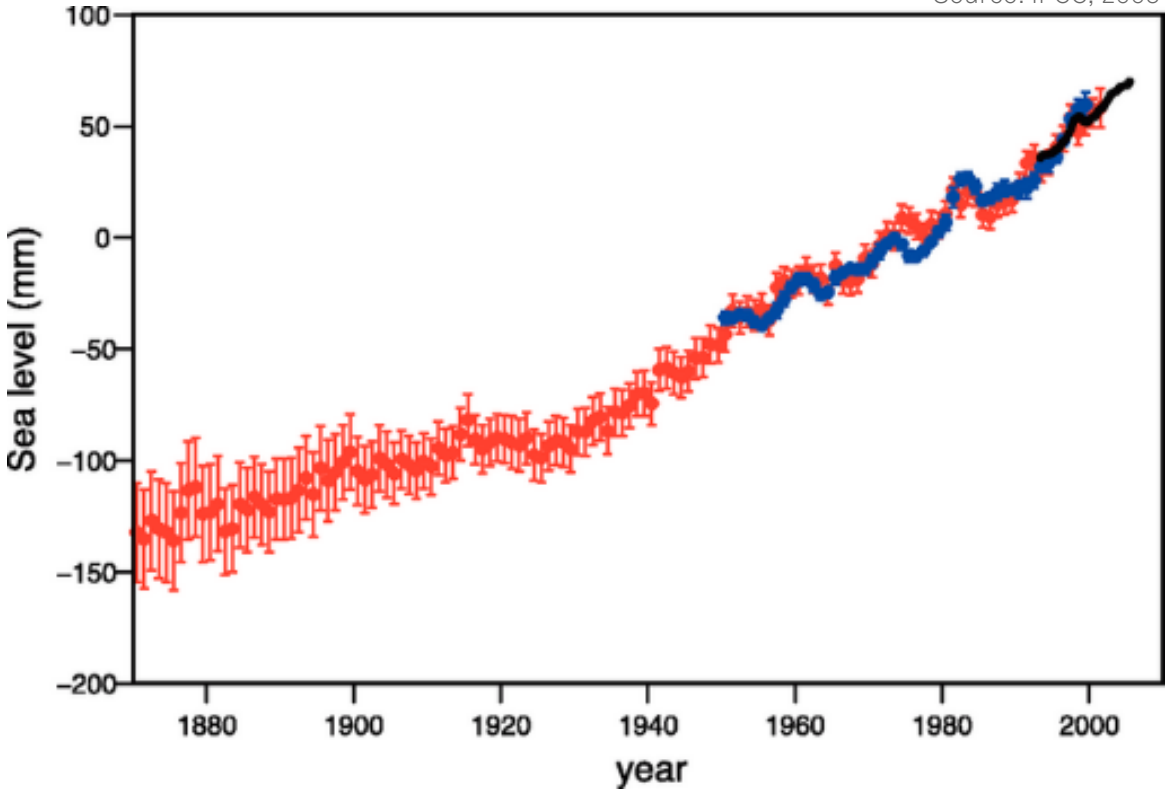
red -- historical records

blue -- tidal gauges

black -- satellite observations

All data show an upward trend!

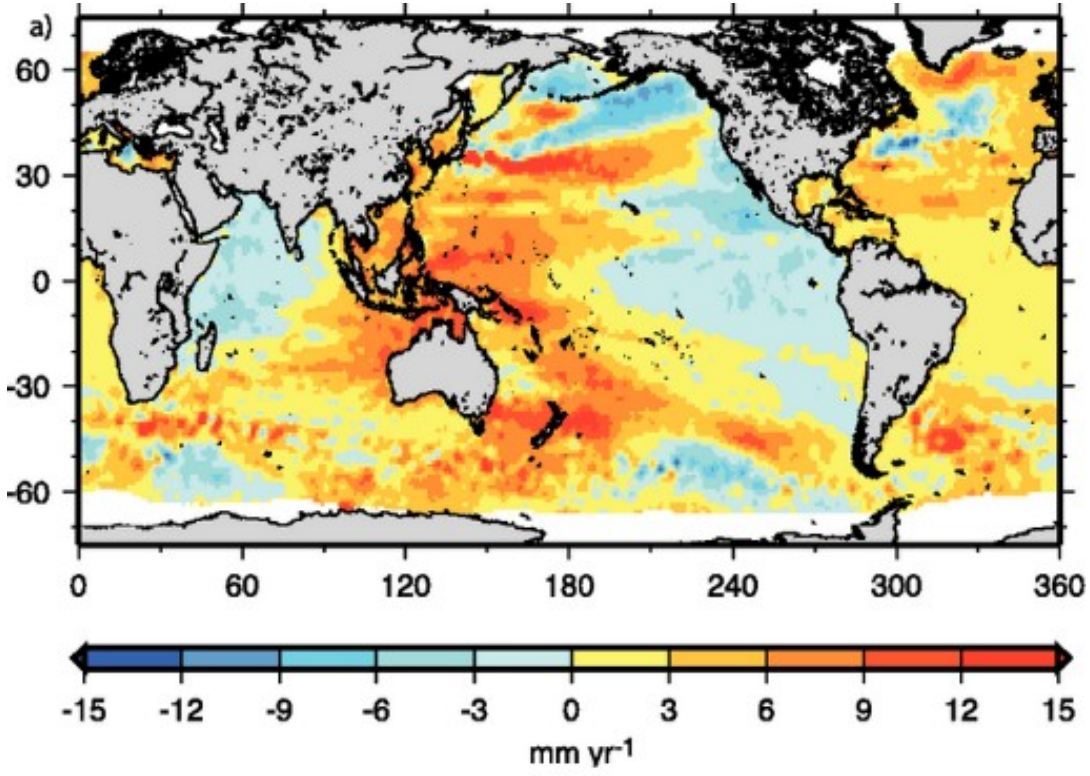
Source: IPCC, 2003



…but the change is not uniform!

Satellite observations show large regional variations in sea-level change.

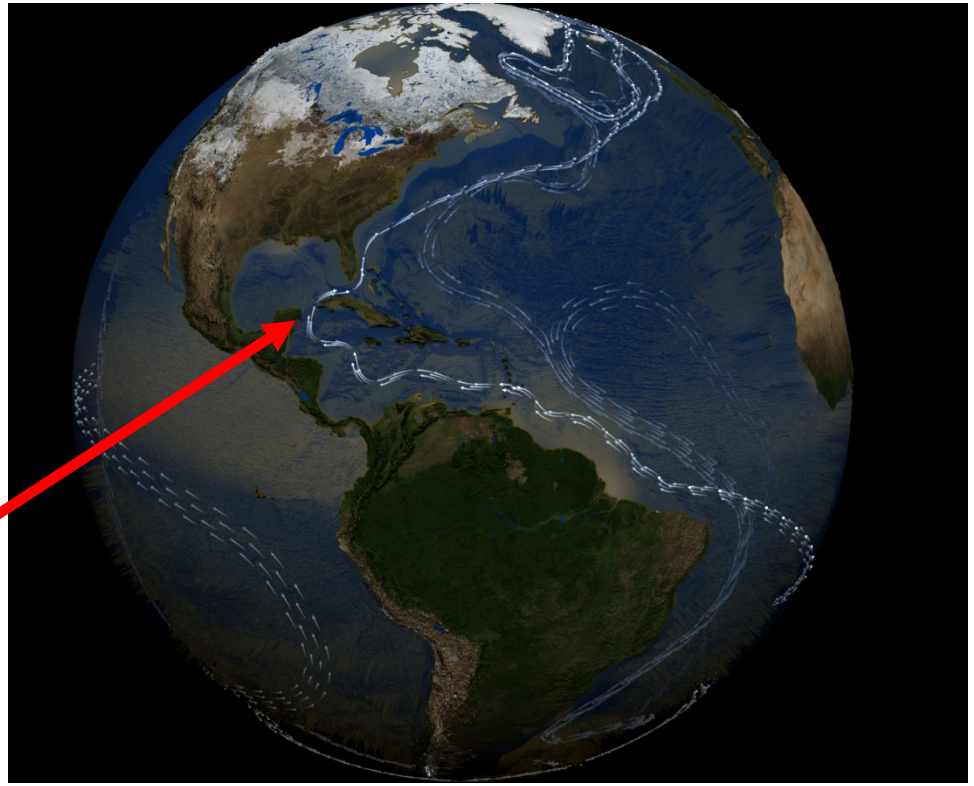
Source: IPCC Sea level trend 1993-2003



Sea level varies regionally

Caused by **large ocean currents** which are driven by temperature, and salinity differences and wind.

Meridional Overturning Circulation (MOC)



Source: NASA/Goddard Space Flight Center Scientific Visualization Studio

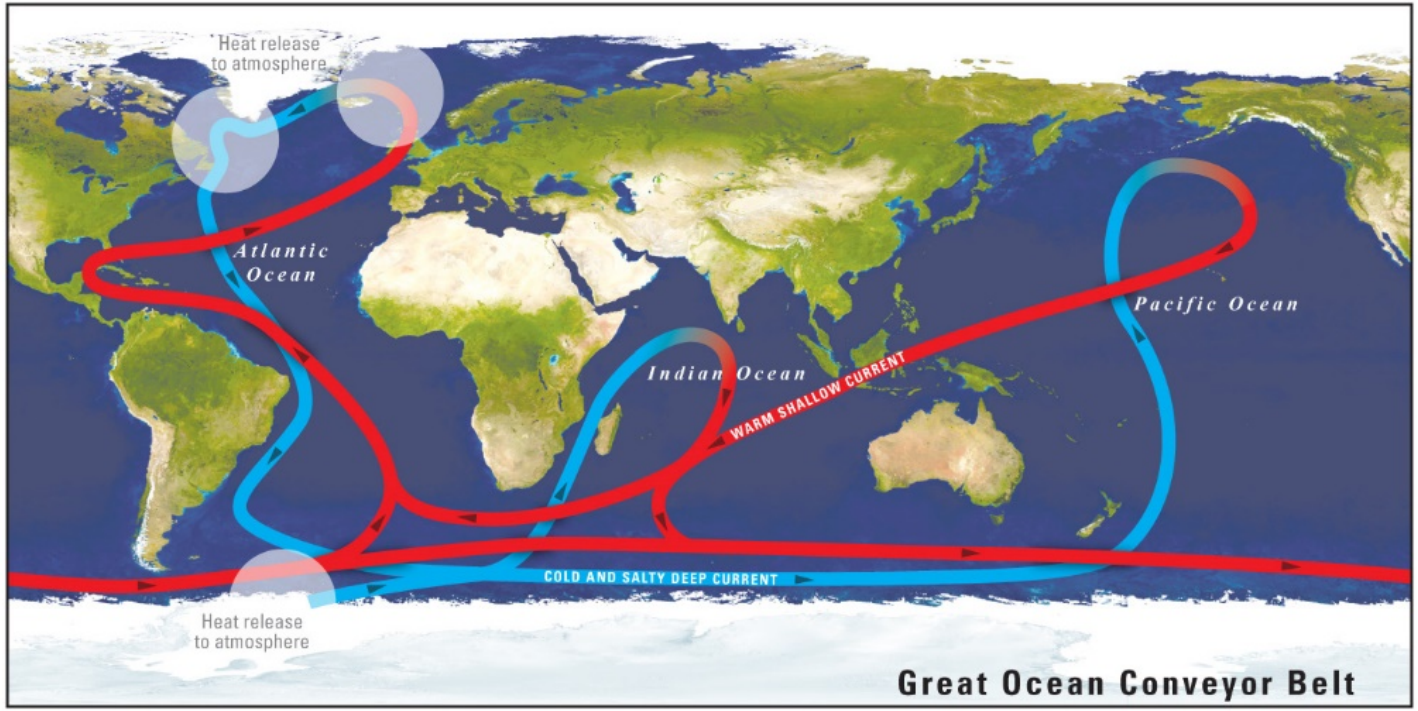
Meridional Overturning Circulation

Water transport:
20 billion liters/sec.

Heat release:
500 GigaWatt

What is the effect of climate change on the MOC?

Use simulations to gain more insight



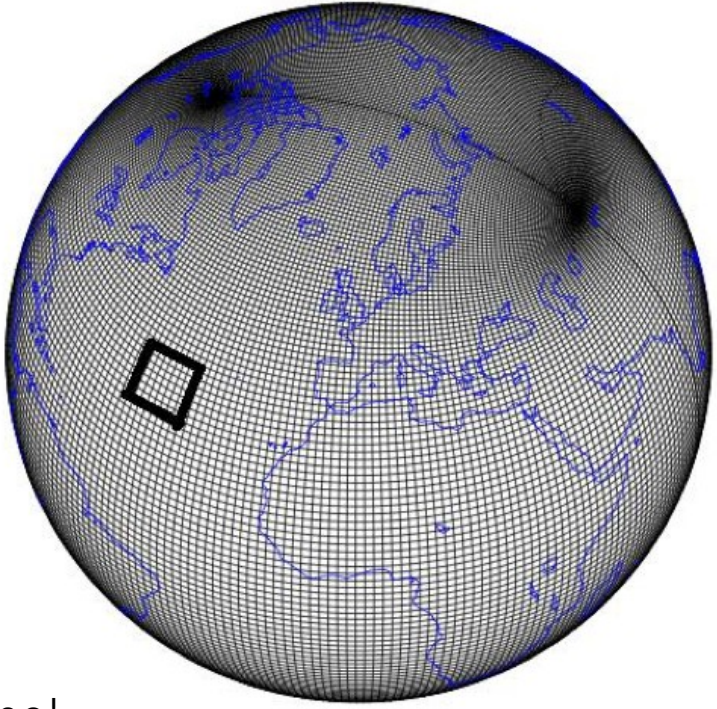
Source: IPCC, 1996

Parallel Ocean Program (POP)

"The POP ocean model is a level-coordinate ocean general circulation model that solves the three-dimensional primitive equations for ocean dynamics"

Resolution is important for the results:
1° resolution (100x100 km) was the norm.
0.1° resolution (10x10 km) is eddie permitting

Direct relation between resolution and compute time!

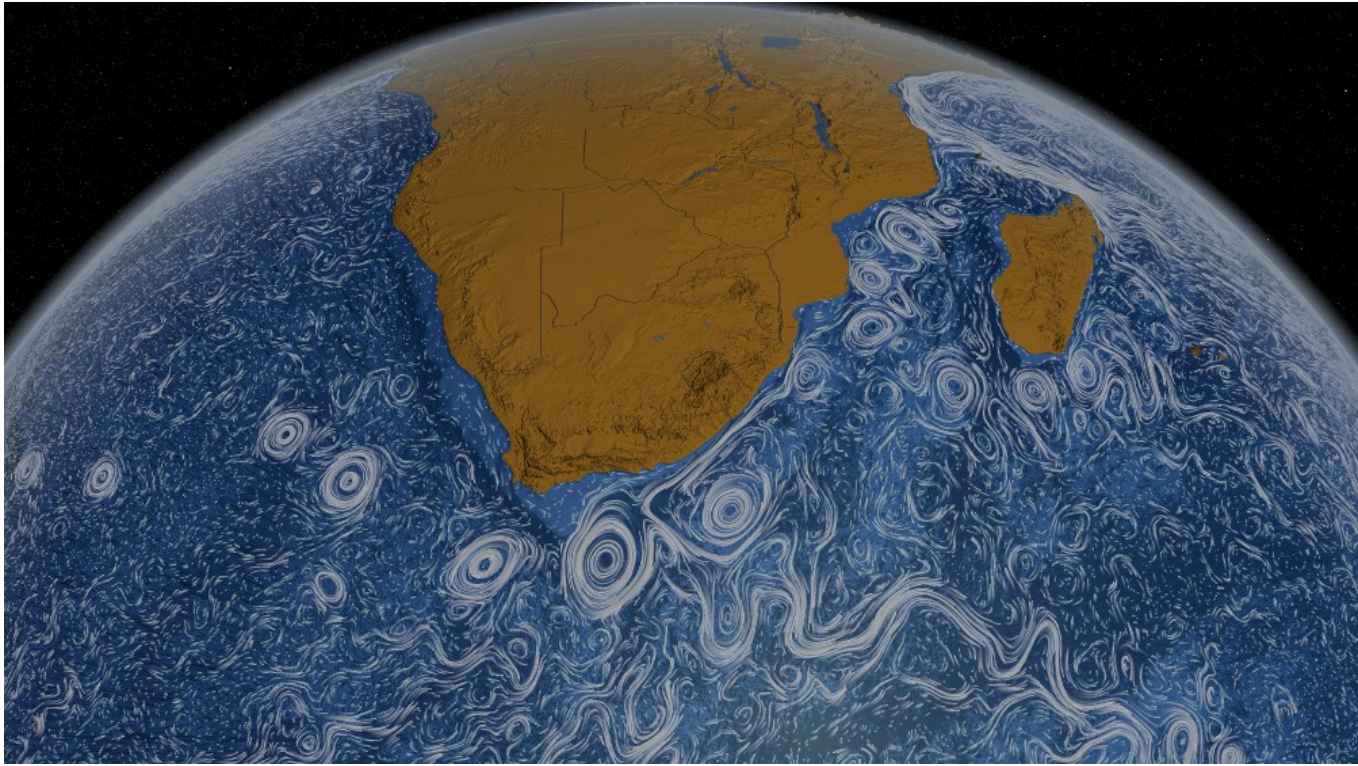


Source: Los Alamos National Laboratory

What are eddies ?

‘Whirlpools’, up to 300 km in diameter and 4 km deep.

They have a large effect on ocean behavior.



Source: NASA/Goddard Space Flight Center Scientific Visualization Studio

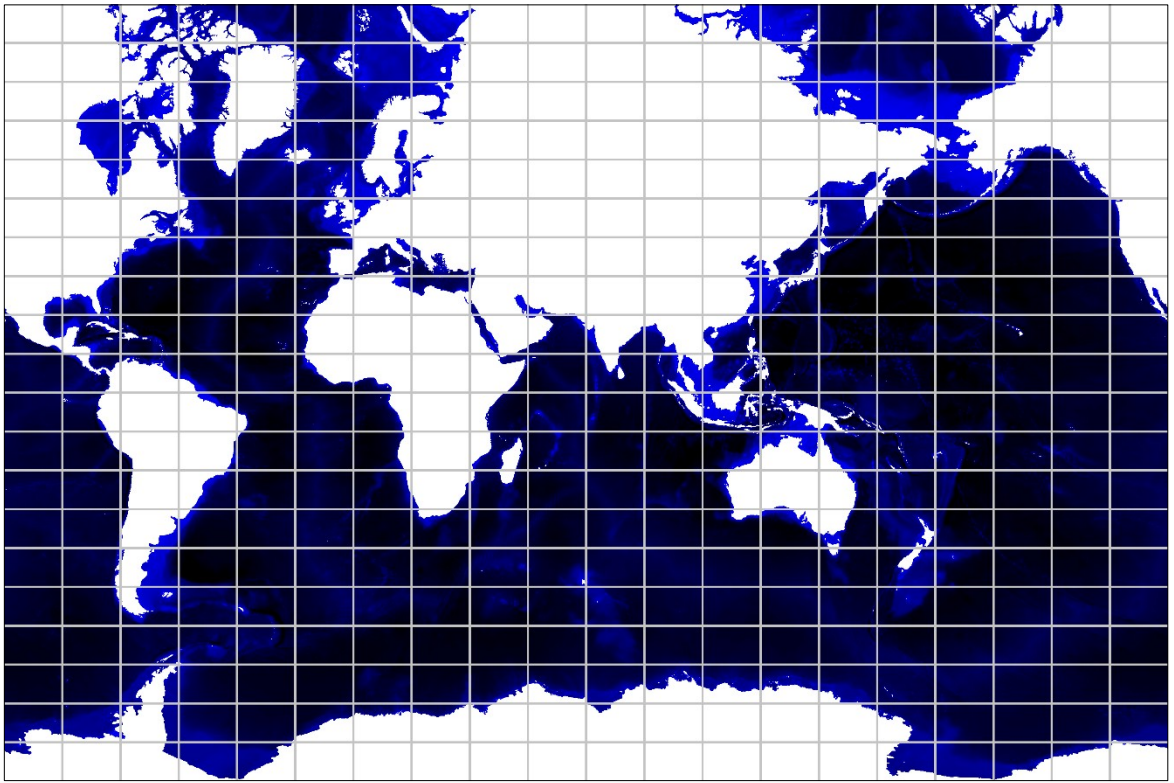
How does POP work ?

Fortran/MPI application (1992)
25 years old!!!

POP divides the world into a grid, which is divided into blocks.

These blocks are distributed over many processes (= cores) using MPI.

Traditionally a cartesian distribution is used that assigns one block to each MPI process.



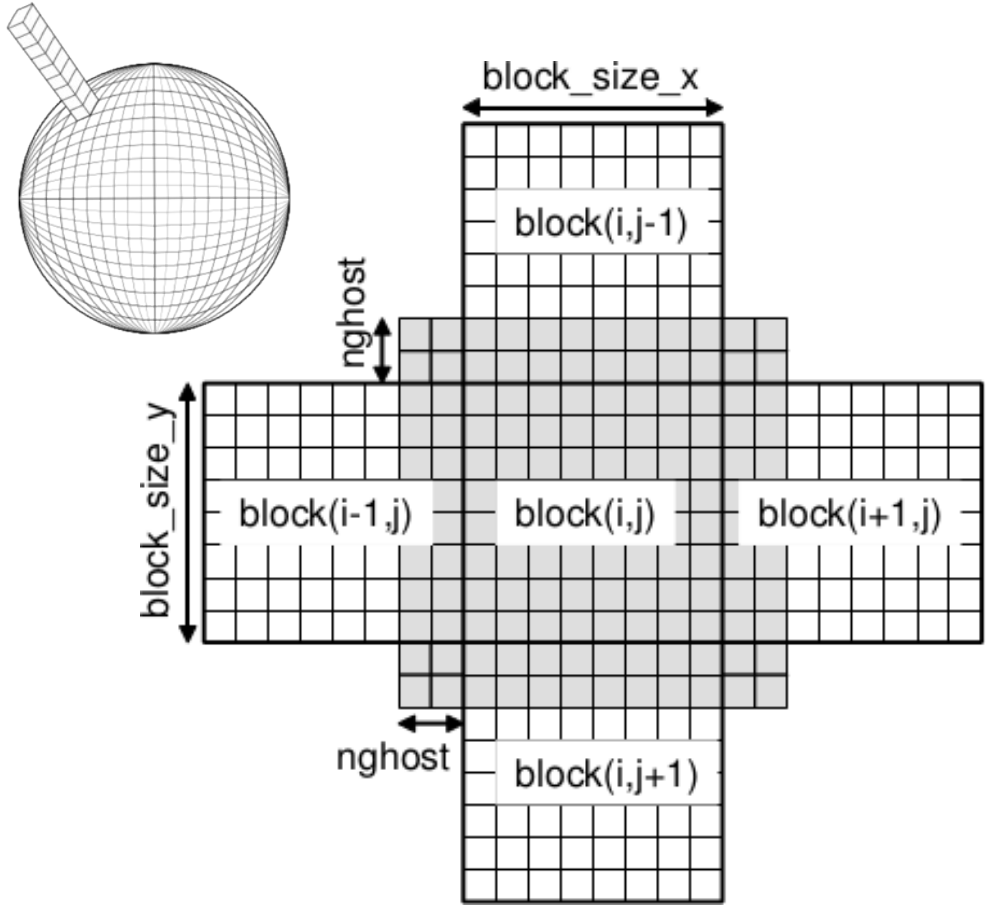
Halo exchange

Each MPI process computes the state of many variables (water temperature, salinity, velocities, etc) in its local 3D column for each time step.

To do this every grid cell needs information from its neighbors.

Therefore, each block needs to exchange data with its neighbors many times during each time step (3D halo exchange).

This is the main source of MPI communication in POP



Source: A Distributed Approach to Improve the Performance of the Parallel Ocean Program
Ben van Werkhoven et al., Geoscientific Model Development, 7, 257-281, 2014

How we run our ocean simulations?

SURFsara Cartesius
40960 cores
117 TB memory
1.0 PFlop/s

Source:SURFSara

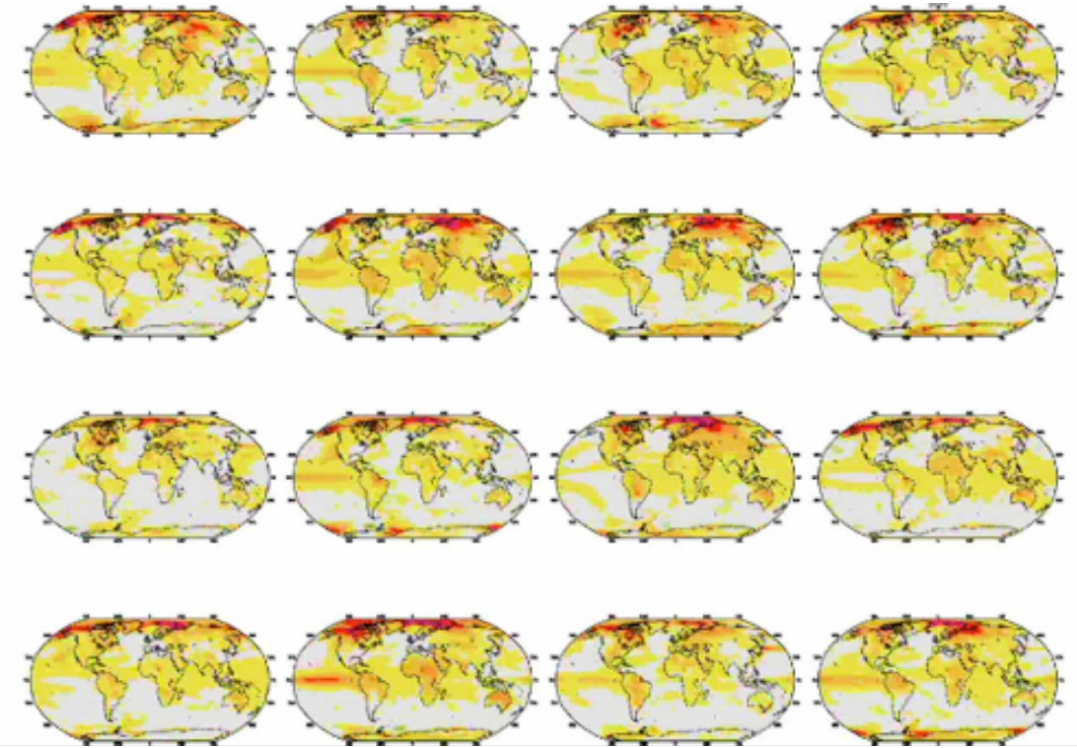
1 simulation of 100 years
at 0.1° resolution (10x10 km)
takes 20 days on O(1000)
cores and produces
10+ TB output.

(but there is more!)

Ensembles

We don’t run 1 simulation but an **ensemble** of 16, each using a slightly different forcing.

16x increase in compute time

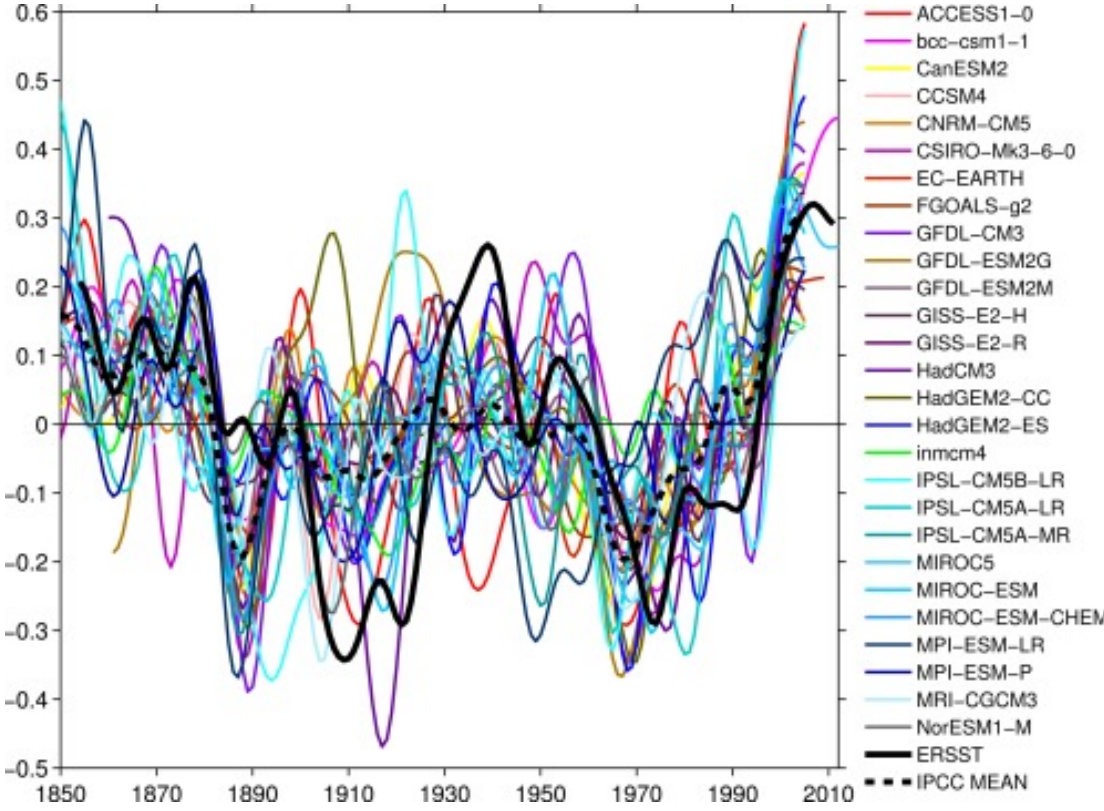


Source: IPCC 1996

Why ensembles?

Climate is a chaotic system:
a small change in forcing, model or starting conditions may change the outcome significantly.

By running many simulations and/or different models, we get many results and do statistics on them to determine the certainty of the results.



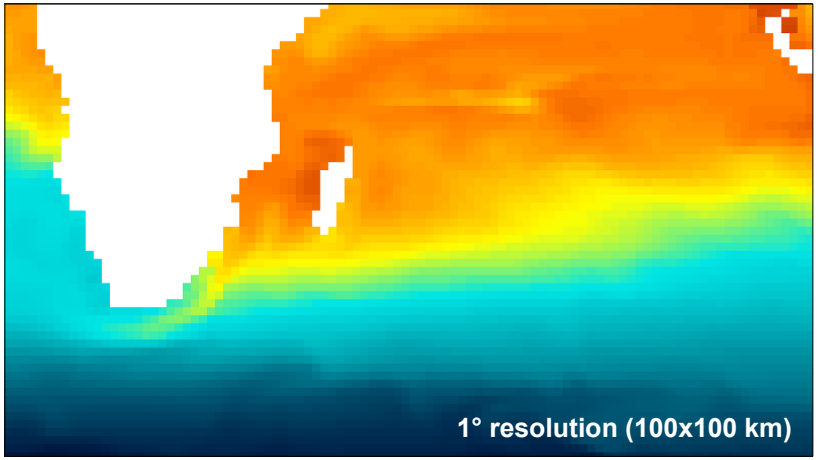
Source: L. Zhang, C. Wang, DOI: 10.1002/jgrc.20390

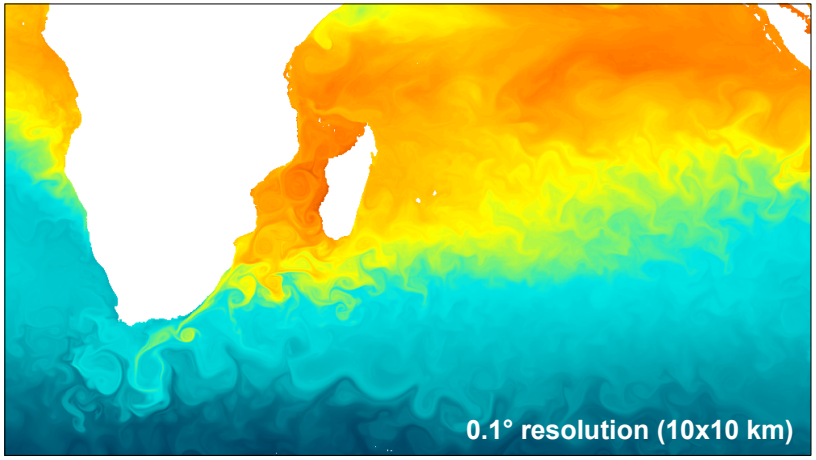
Higher Resolution

0.1° resolution (10x10 km) is only the start! We want to increase the model resolution even further to get more detailed results.

Ultimate goal (last time I asked):
0.01° resolution (1x1 km)
(fully eddie resolving)

100x increase in compute time!





Source: eSalsa results

We need more compute power

Needed for (a combination of):
higher resolution + large ensembles

We looked at the following solutions:

Distributed task farming - for running ensembles
Accelerators - for more compute power per node
Distributed computing - for more overall compute power

Visualization - for fast evaluation of the results

distributed computing

(the simple way)

Distributed Task Farming

Running ensembles takes a long time and quite a bit of effort!

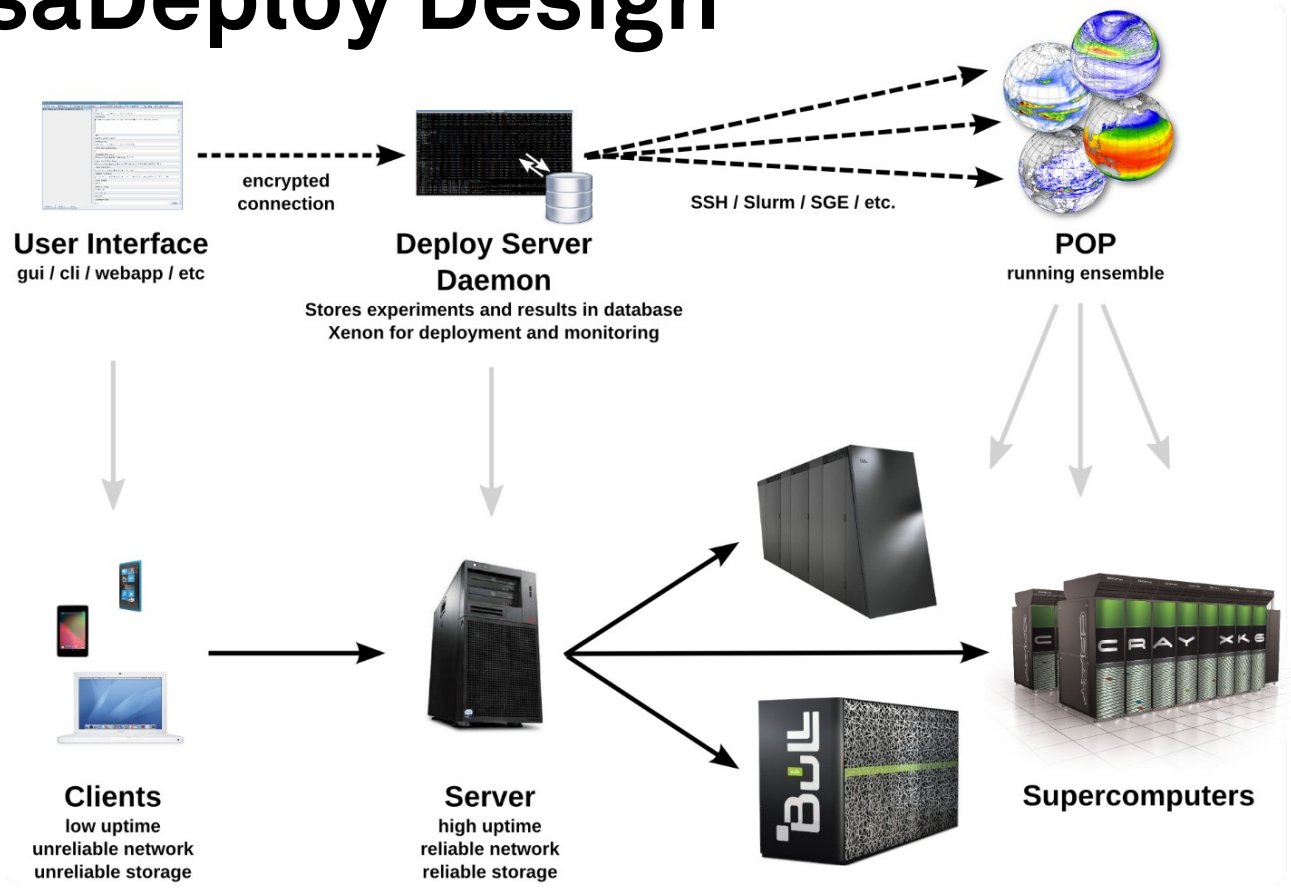
We need to:

- create submission scripts for each target supercomputer
- (possibly) adapt submission scripts for each ensemble member
- create POP configurations for each ensemble member
- need to 'babysit' each ensemble run (which may take up to a month).

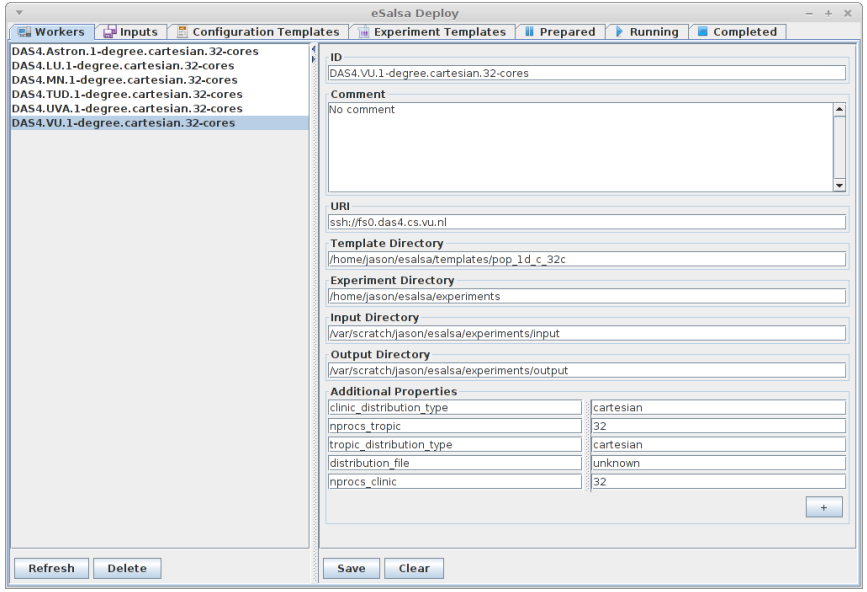
eSalsaDeploy help the scientist by:

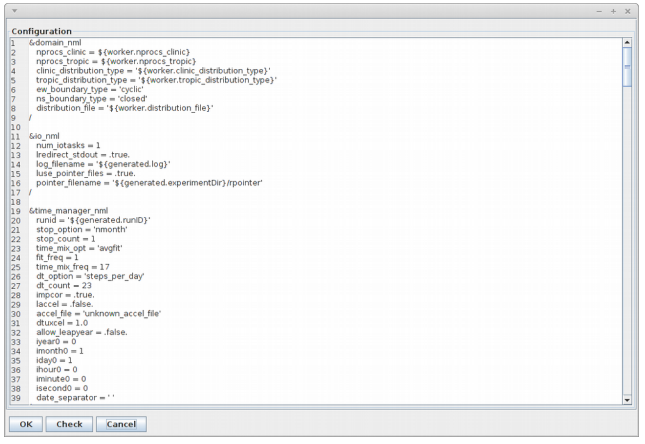
- automatically running ensemble members on multiple supercomputers
- generating the necessary config files
- managing the input files, logs files and results.
- babysit the application

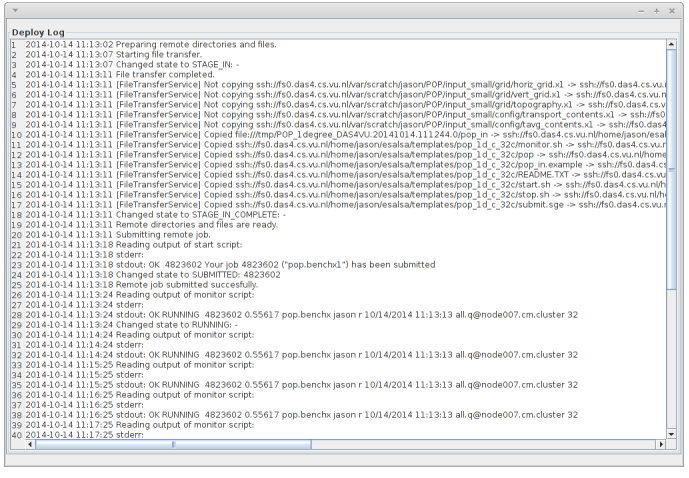
eSalsaDeploy Design



eSalsaDeploy GUI

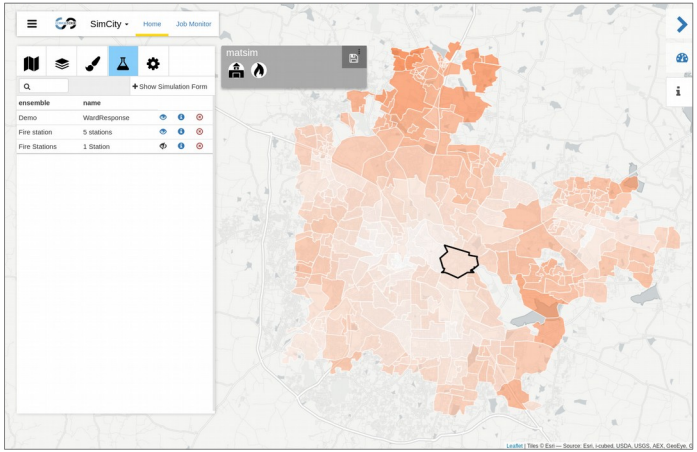




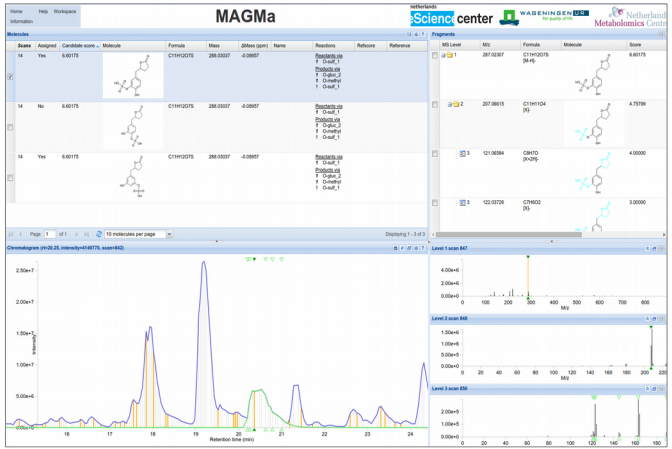


Edit machine descriptions, configuration and experiment templates, remotely start experiments, inspect log files, etc...

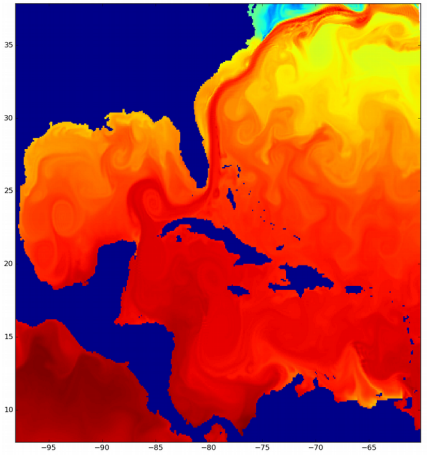
Compute at the touch of a button



SIMCITY



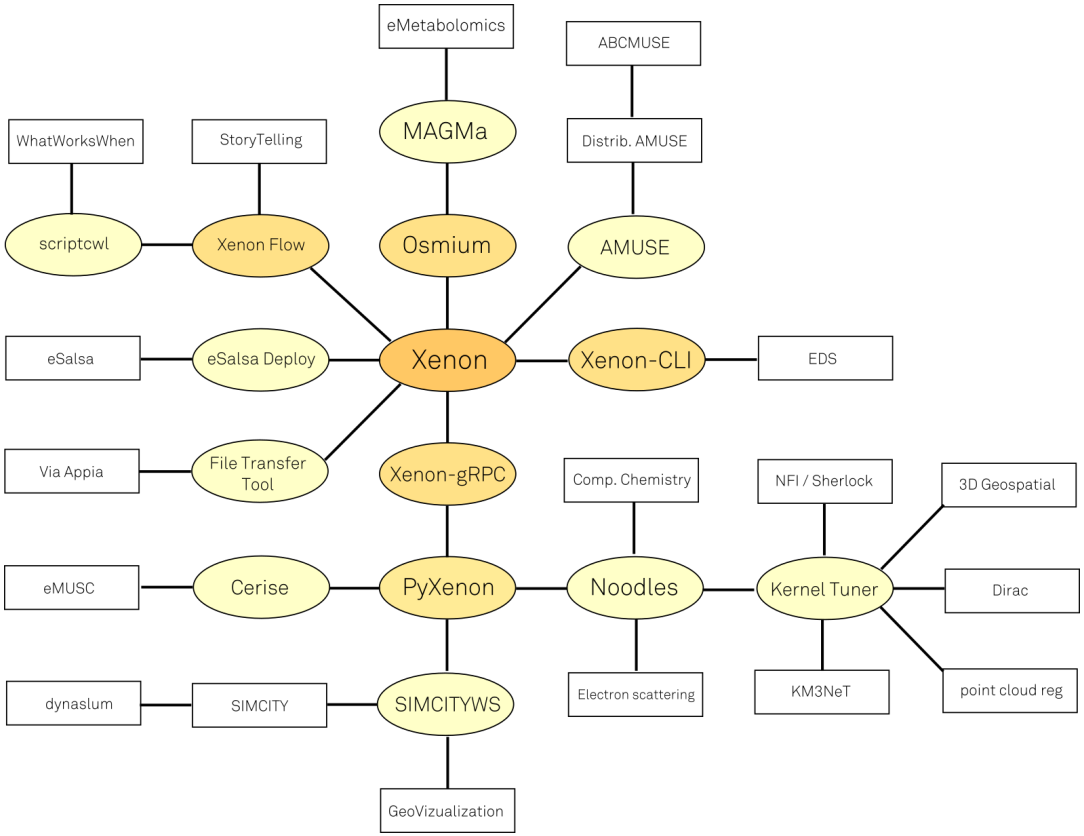
MAGMa



OMUSE

This “deployment and coupling” turned out to be a recurring theme in many of our projects: easy access to compute and storage and babysitting applications

Xenon and friends



Xenon is a software library that provides easy access to compute and storage.

It is currently the used in 14 other tools, which are used in 19 different projects.

This makes generic tools like Xenon more sustainable, as we spread their development efforts over many different projects.

Descendant from JavaGAT developed here at the VU!

eScience center

accelerators

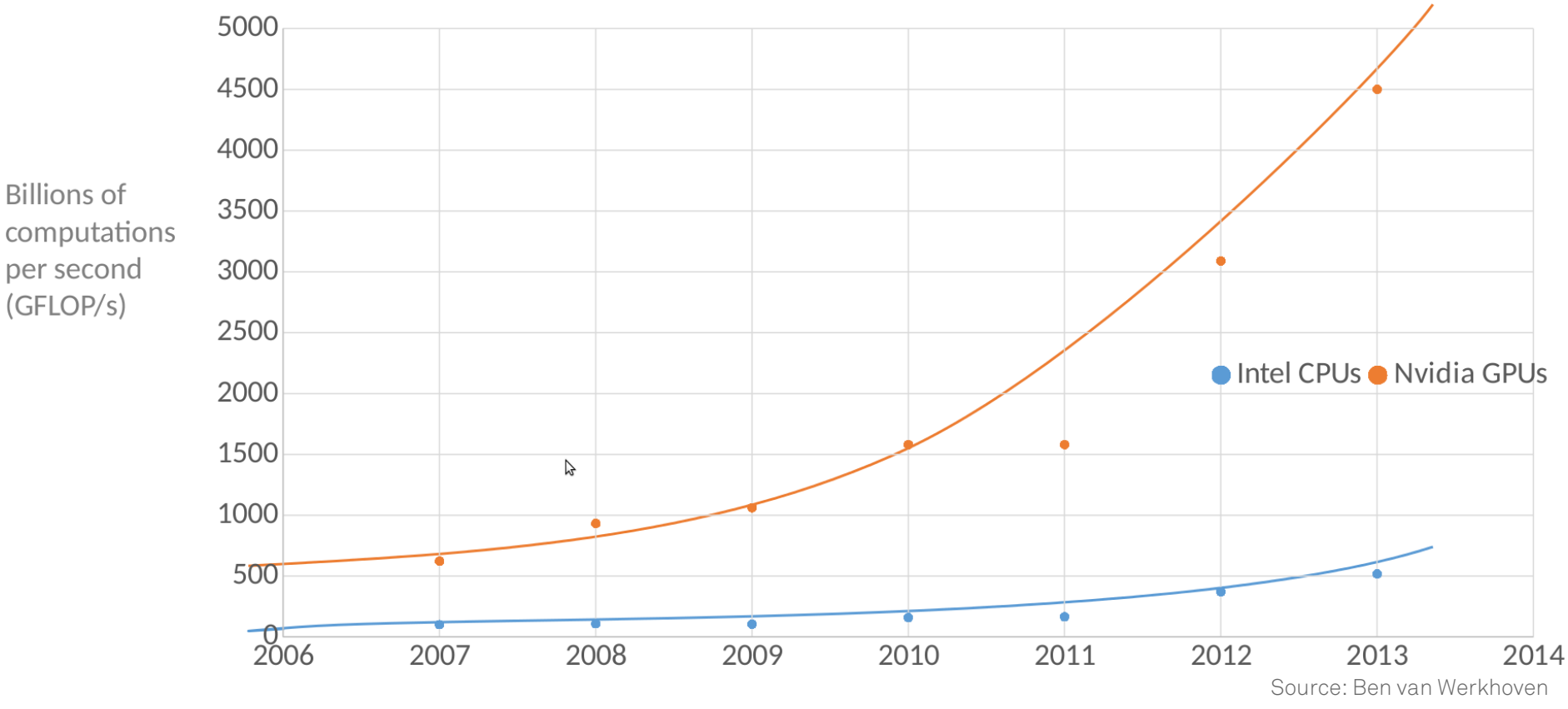
Accelerators

Accelerators (such as GPUs) offer a huge increase in compute power.

NVidia P100 PCI version
3584 cores, 4.7 Tflop/s
300 Watt
Fastest Nvidia GPU available in 2017
$6000

ASCI Red
9632 cores, 2.4 Tflop/s
850.000 Watt
Fastest super computer in the world 1997-2000
$46.000.000

Performance of GPUs and CPUs

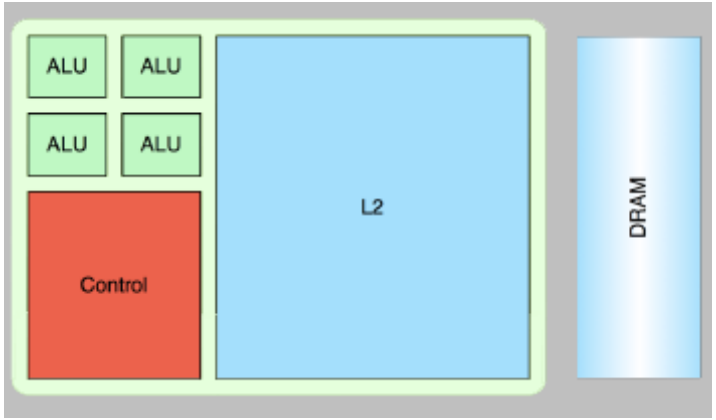


Source: Ben van Werkhoven

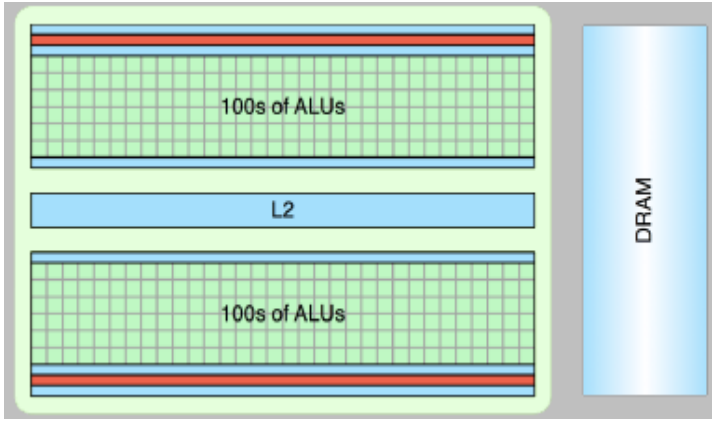
GPUs

Massively parallel: work is divided across thousands of extremely light-weight threads (usually one thread per element)

Functions that run on the GPU are called kernels



Typical CPU vs GPU design

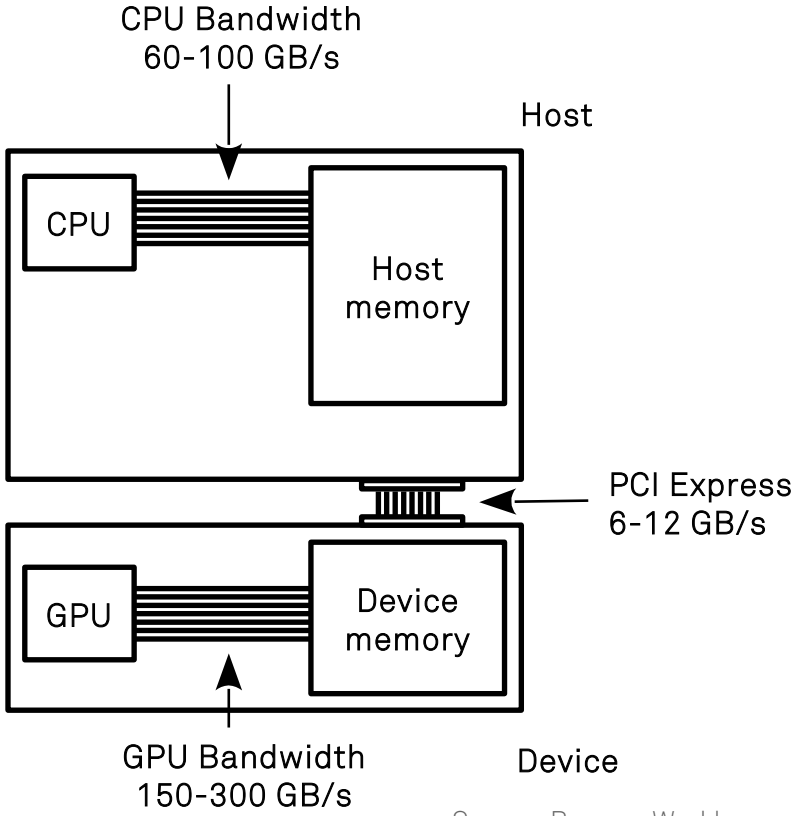


GPUs

Requires special programming languages (CUDA, OpenCL)

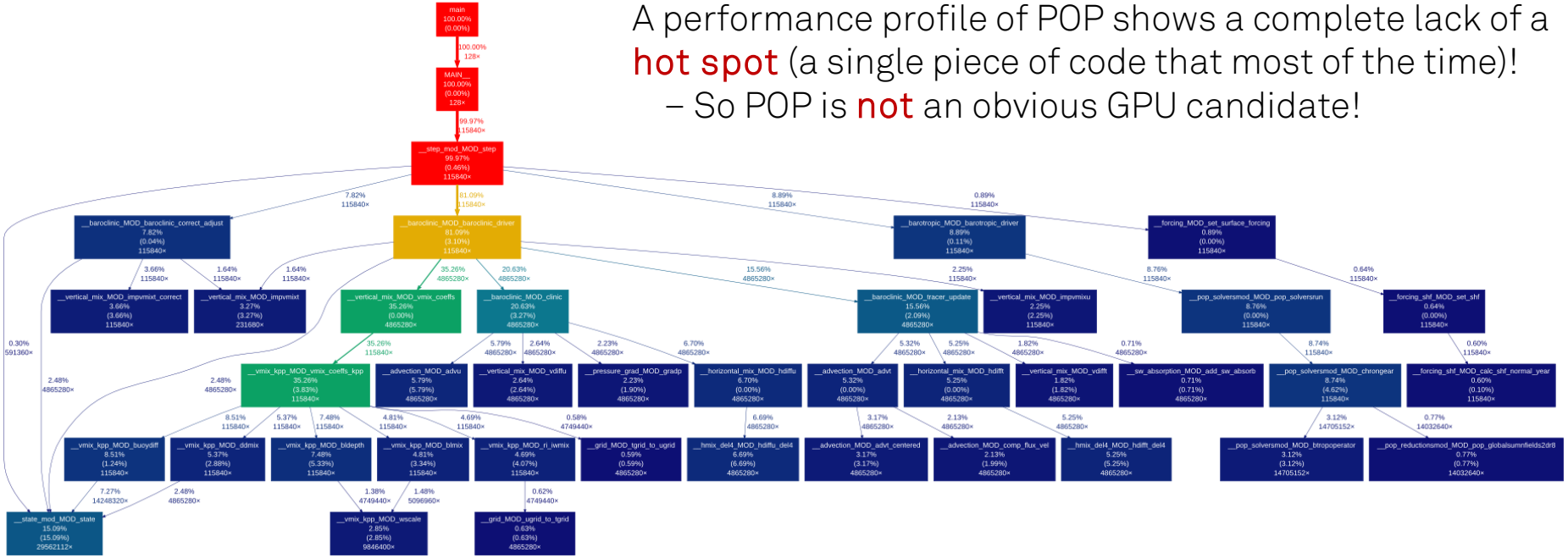
Separate device in computer

Transferring data from the CPU memory to GPU memory is part of the application and can be a major bottleneck!



Source: Ben van Werkhoven

Is POP suitable for GPUs ?

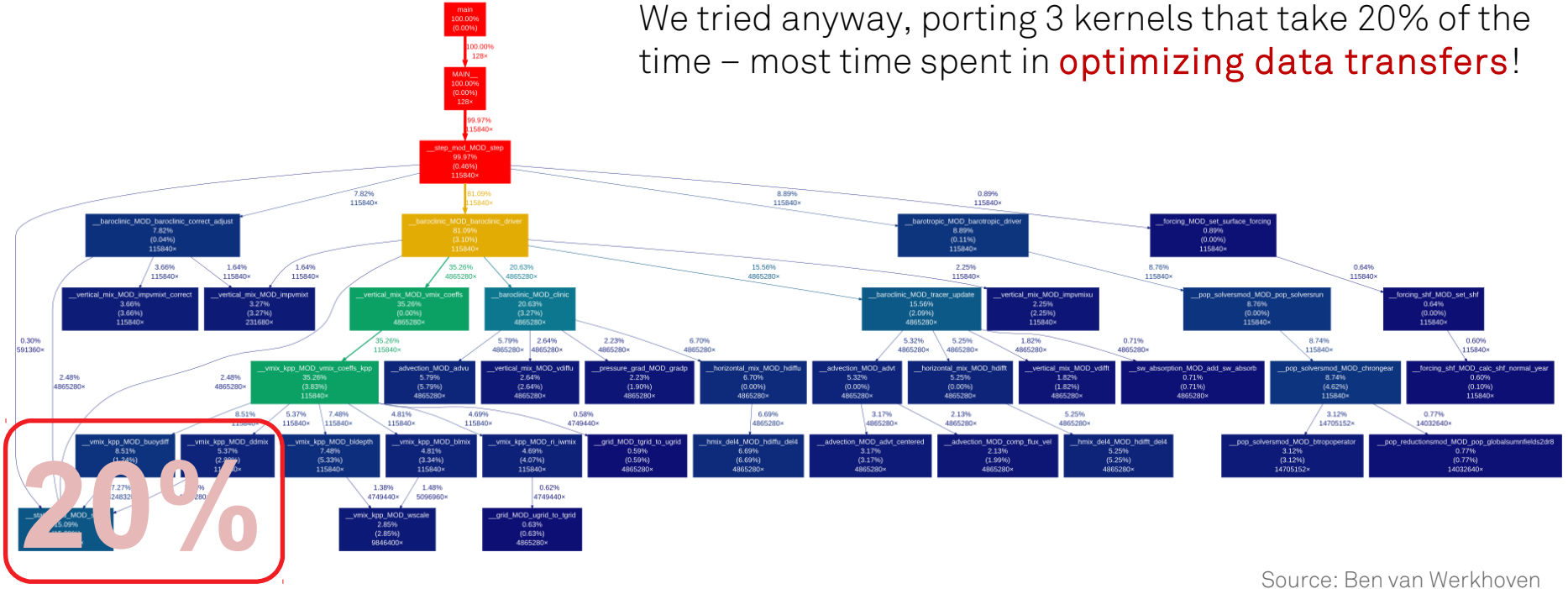


A performance profile of POP shows a complete lack of a hot spot (a single piece of code that most of the time)!

– So POP is not an obvious GPU candidate!

Source: Ben van Werkhoven

Is POP suitable for GPUs ?



20%

We tried anyway, porting 3 kernels that take 20% of the time – most time spent in optimizing data transfers!

Source: Ben van Werkhoven

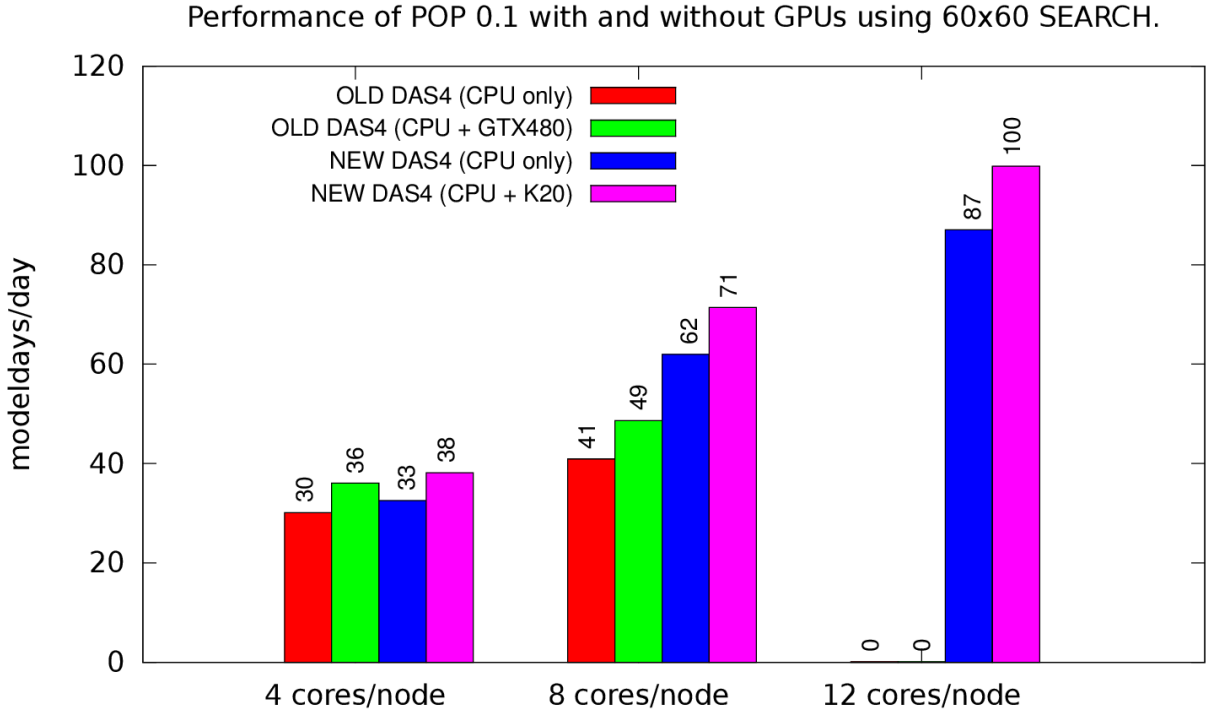
Ported 20% of code to CUDA

Initial results

Porting 20% of POP to GPUs, improves performance with 13%

Sharing 1 GPU with 12 CPU cores is not a problem: the GPU has flops to spare!

CPU-GPU data transfer is the bottleneck, not compute!



Source: A Distributed Approach to Improve the Performance of the Parallel Ocean Program
Ben van Werkhoven et al., Geoscientific Model Development, 7, 257-281, 2014

Performance Models

This research also lead to the paper:

Performance Models for CPU-GPU data transfers,
Ben van Werkhoven et al, CCGrid 2014
(best paper nominee)

Source: Performance Models for CPU – GPU Data Transfers, Ben van Werkhoven et al., 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), best paper nominee, Chicago, IL, May 2014.

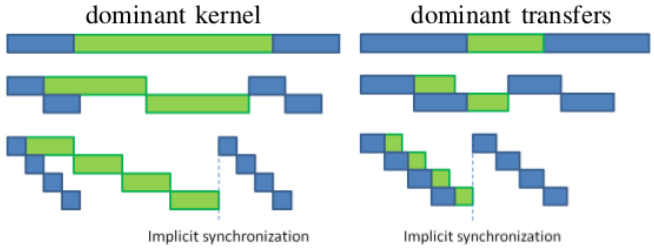


Fig. 1. Diagrams showing possible overlap for devices with implicit synchronization and 1 copy engine when using 1, 2, or 4 streams.

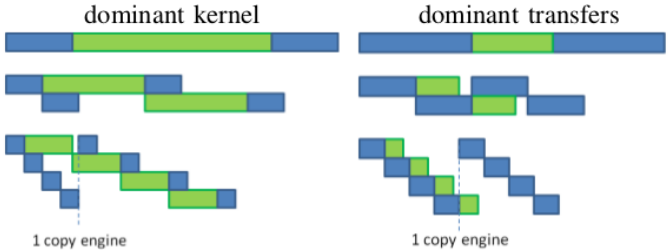


Fig. 2. Diagrams showing possible overlap for devices with no implicit synchronization and 1 copy engine when using 1, 2, or 4 streams.

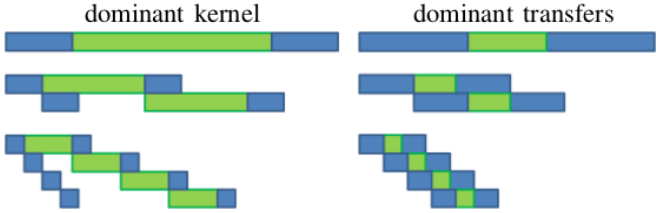


Fig. 3. Diagrams showing possible overlap for devices with no implicit synchronization and 2 copy engines when using 1, 2, or 4 streams.

Performance Models

Ben's performance models can be used to estimate the performance of each of the data transfer options.

Implicit synchronization and 1 copy engine.

Dominant kernel

$$time = L + o + \frac{B_{hd}}{nStreams} \times G_{hd} + t_E + L + o + B_{dh} \times G_{dh} + g \times (nStreams - 1) \quad (1)$$

Dominant transfers

$$time = L + o + B_{hd} \times G_{hd} + g \times (nStreams - 1) + \frac{t_E}{nStreams} + L + o + B_{dh} \times G_{dh} + g \times (nStreams - 1) \quad (2)$$

No implicit synchronization and 1 copy engine.

Dominant kernel

$$time = L + o + \frac{B_{hd}}{nStreams} \times G_{hd} + t_E + L + o + \frac{B_{dh}}{nStreams} \times G_{dh} \quad (3)$$

Dominant transfers

$$\begin{aligned} time = max(&L + o + B_{hd} \times G_{hd} + g \times (nStreams - 1) + L + o + B_{dh} \times G_{dh} + g \times (nStreams - 1), \\ &L + o + B_{hd} \times G_{hd} + g \times (nStreams - 1) + \frac{t_E}{nStreams} + L + o + \frac{B_{dh}}{nStreams} \times G_{dh}, \\ &L + o + \frac{B_{hd}}{nStreams} \times G_{hd} + \frac{t_E}{nStreams} + L + o + B_{dh} \times G_{dh} + g \times (nStreams - 1)) \end{aligned} \quad (4)$$

No implicit synchronization and 2 copy engines.

$$\begin{aligned} time = max(&L + o + B_{hd} \times G_{hd} + g \times (nStreams - 1) + \frac{t_E}{nStreams} + L + o + \frac{B_{dh}}{nStreams} \times G_{dh}, \\ &L + o + \frac{B_{hd}}{nStreams} \times G_{hd} + t_E + L + o + \frac{B_{dh}}{nStreams} \times G_{dh}, \\ &L + o + \frac{B_{hd}}{nStreams} \times G_{hd} + \frac{t_E}{nStreams} + L + o + B_{dh} \times G_{dh} + g \times (nStreams - 1)) \end{aligned} \quad (5)$$

Device-mapped host memory.

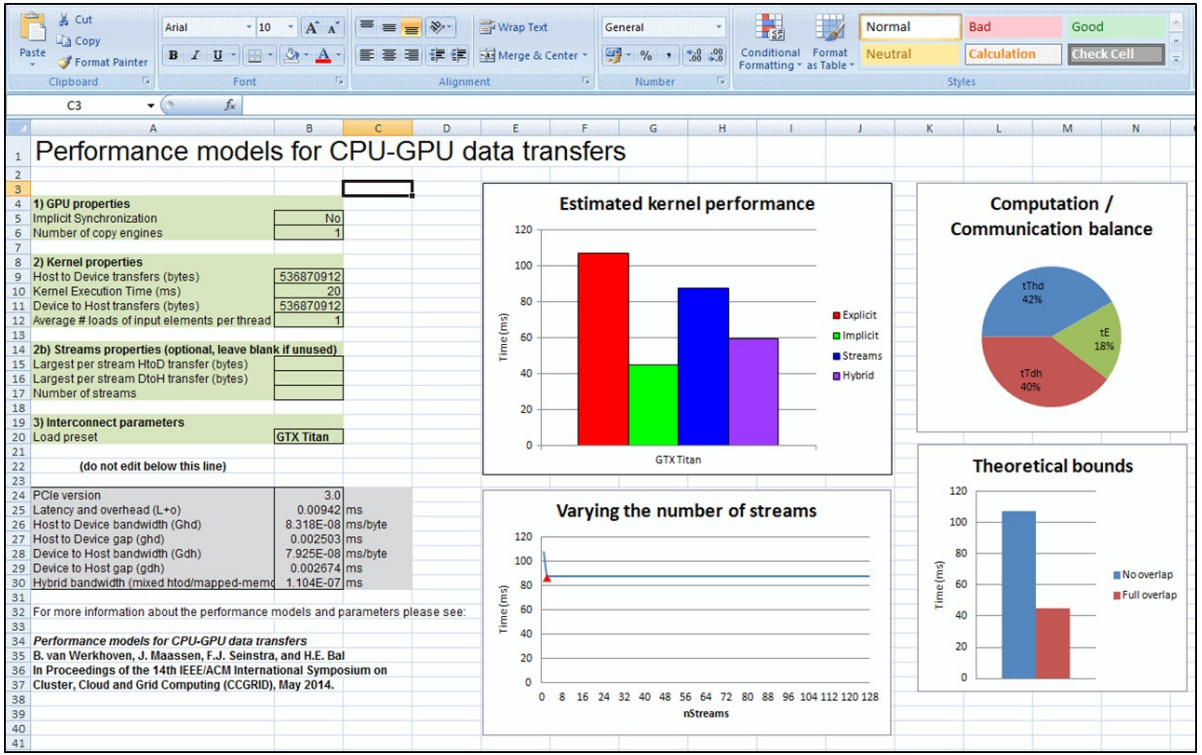
$$time = max(L + o + B_{hd} \times G_{hd} + L + o, L + o + t_E + L + o, L + o + L + o + B_{dh} \times G_{dh}) \quad (6)$$

Fig. 7. Overview of performance models. B_{hd} is the number of bytes transferred from host to device. B_{dh} is the number of bytes transferred from device to host.

Source: Ben van Werkhoven

Performance Models

Using a simple excel sheet
You can test what the optimal transfer method is for your code!



Source: Ben van Werkhoven

Performance Models

Results show that the performance estimates are quite accurate.

Estimates for implicit implementation are most accurate (max error 3.85%).

Estimates for hybrid implementation have largest error (max error 10.75%).

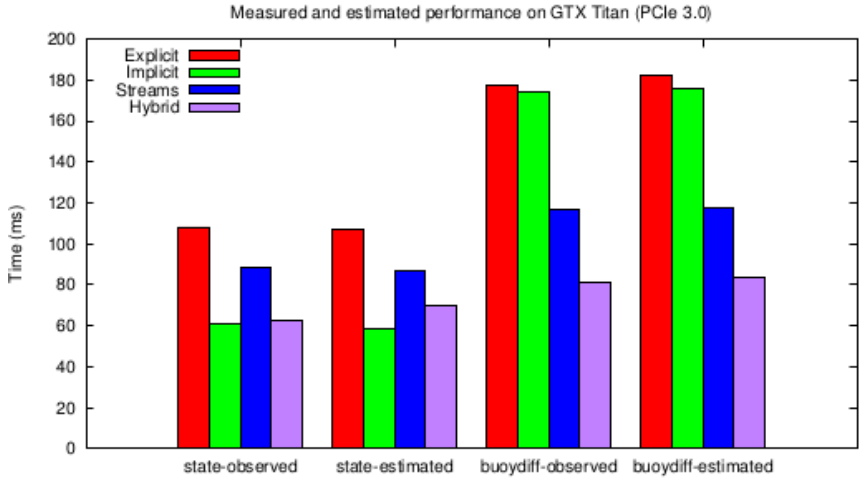


Fig. 9. Performance of state and buoydiff kernels on GTX Titan.

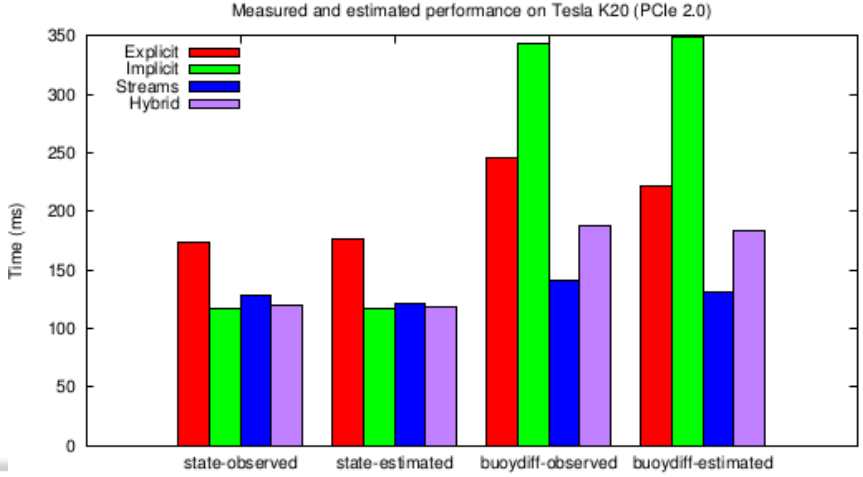


Fig. 10. Performance of state and buoydiff kernels on Tesla K20.

Source: Ben van Werkhoven

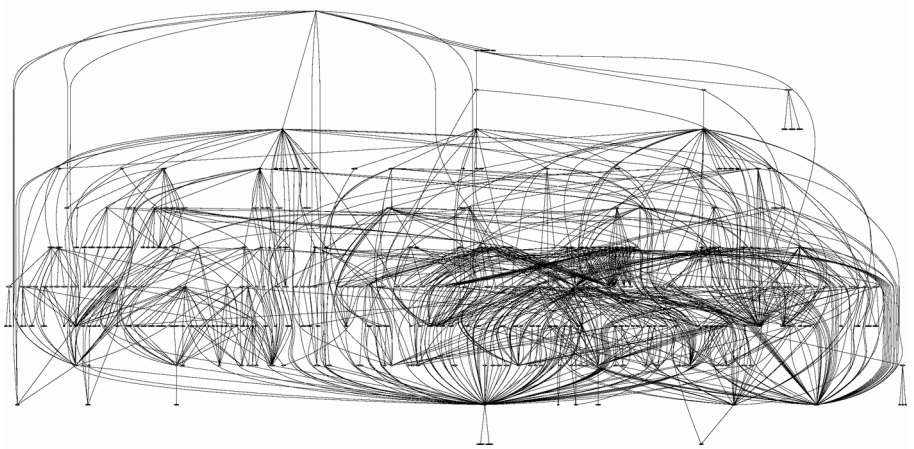
The joy of porting...

Porting 20% of POP to CUDA is hard!

It is hard get insight into the inner workings of a Fortran application that has been worked on for 25 years!

In addition, a lot of 'boilerplate code' is required to connect the Fortran and CUDA parts.

We need more powerful tools!

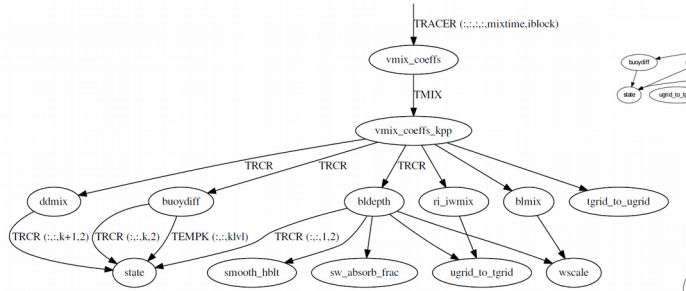


Callgraph of the POP

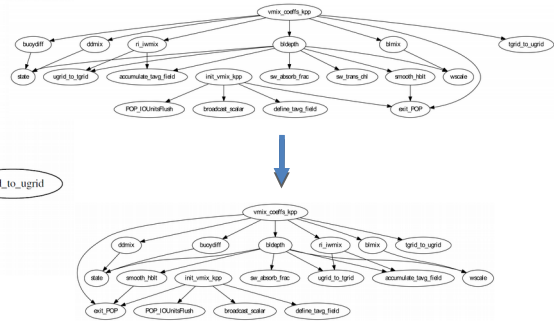
Source: Ben van Werkhoven

Marver: semi-automatic translation

source code analysis



transformation



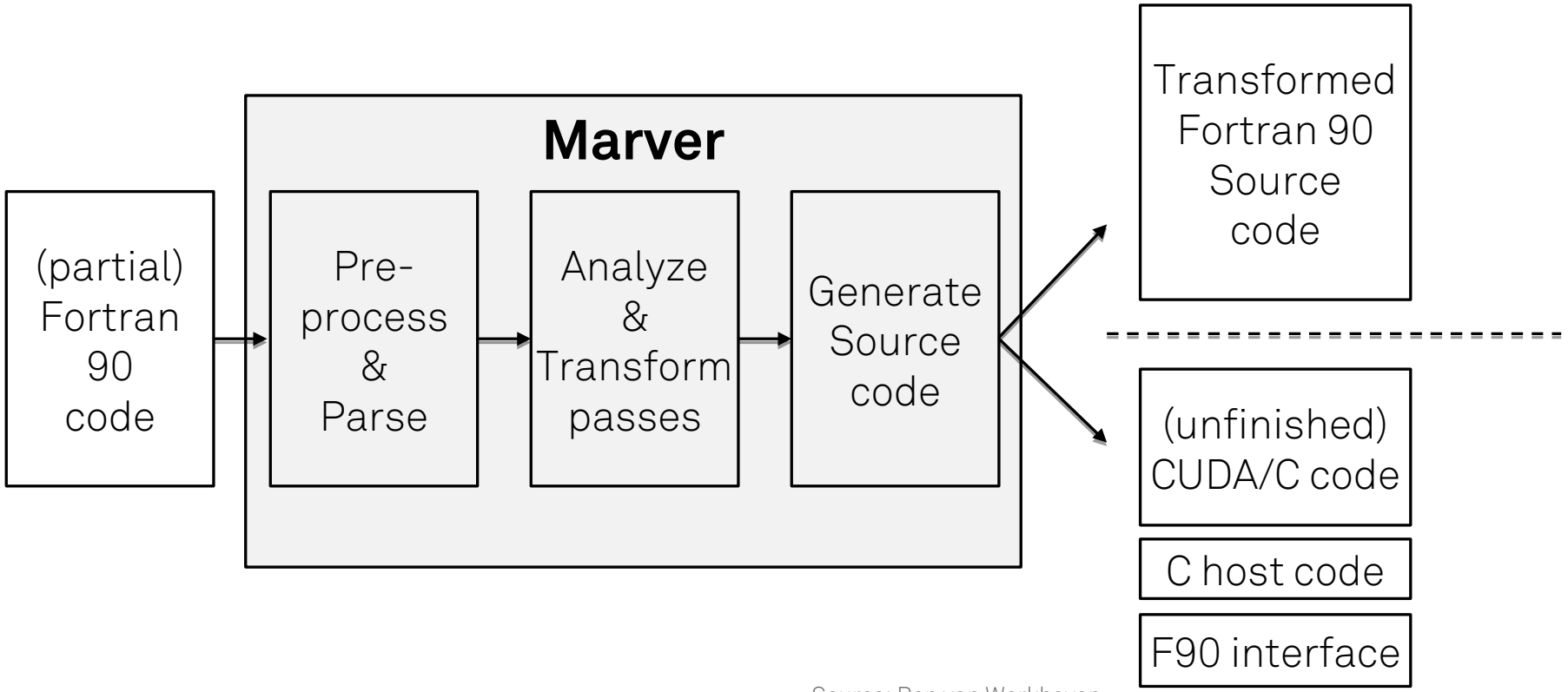
source-to-source translation

```
VVC(:,:,k) = merge(WORK2, c0,
                   (k < KMU(:,:,bid)))
```

```
VVC(i,j,k) = ((k < KMU(i,j,bid) ) ?
                WORK2 : c0);
```

Source: Ben van Werkhoven

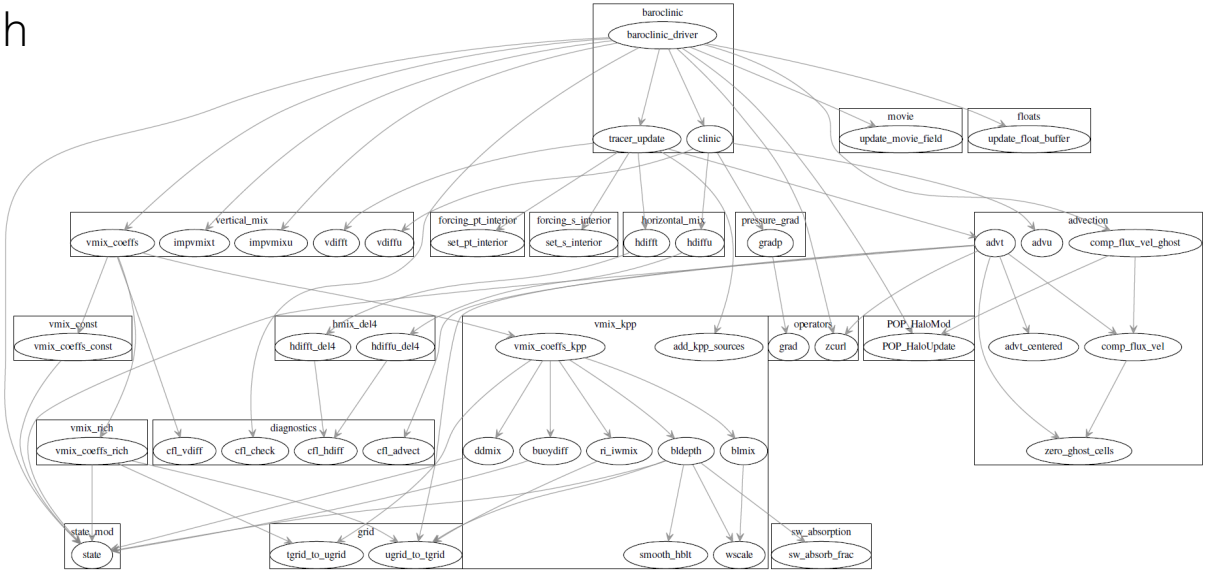
Marver workflow



Source: Ben van Werkhoven

Marver is semi-automatic

Marver generates a call graph

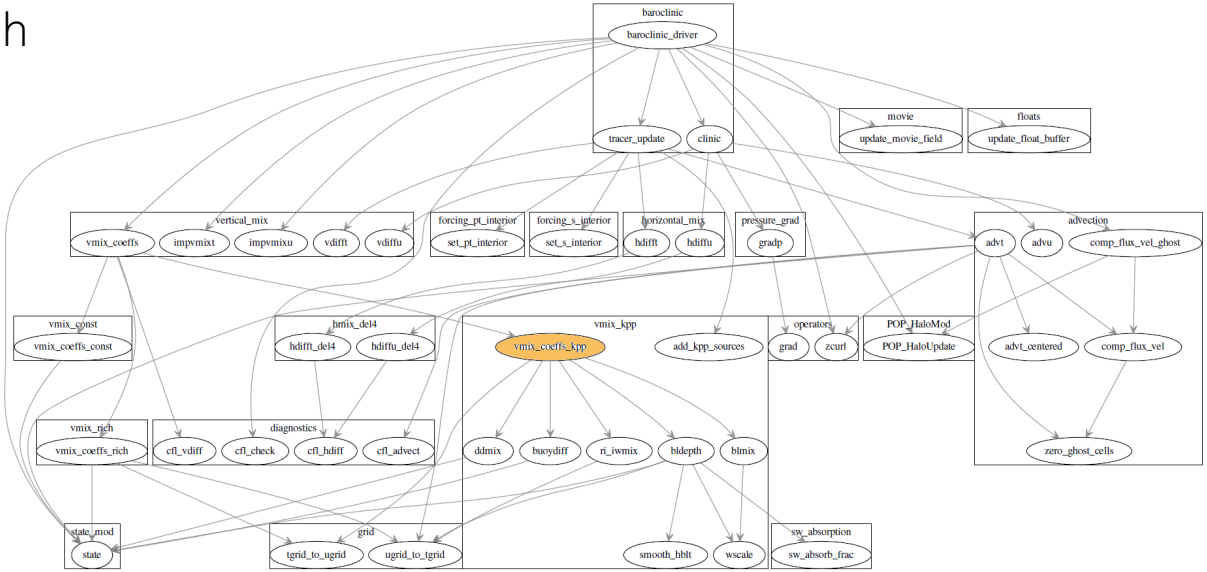


Source: Ben van Werkhoven

Marver is semi-automatic

Marver generates a call graph

User needs to identify entry point in call-graph and parallelism



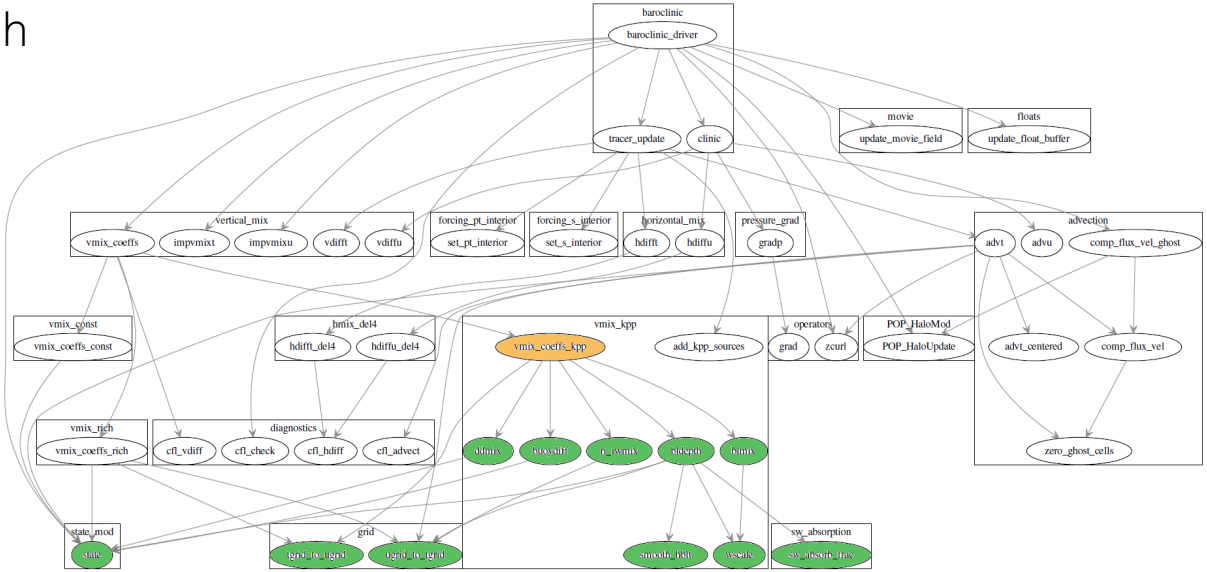
Source: Ben van Werkhoven

Marver is semi-automatic

Marver generates a call graph

User needs to identify entry point in call-graph and parallelism

Marver traverses the call graph and generates CUDA and boilerplate code



Source: Ben van Werkhoven

Marver produces human readable code

User needs to finish the CUDA manually

It is important that Marver generates human readable CUDA code than has a clear link to the original Fortran

Fortran 90 of code:

```
do j=1,ny_block
do i=1,nx_block
   if ( k >= KBL(i,j) ) then
      VISC(i,j,k)   = VISC(i,j,k)   + convect_visc * FCON(i,j)
      VDC(i,j,k,1) = VDC(i,j,k,1) + convect_diff * FCON(i,j)
      VDC(i,j,k,2) = VDC(i,j,k,2) + convect_diff * FCON(i,j)
   endif
end do
end do
```

CUDA/C kernel code:

```
if ( k >= KBL(i,j)-1 ) {
   VISC(i,j,k)   = VISC(i,j,k)   + convect_visc * FCON;
   VDC(i,j,k,0) = VDC(i,j,k,0) + convect_diff * FCON;
   VDC(i,j,k,1) = VDC(i,j,k,1) + convect_diff * FCON;
}
```

Source: Ben van Werkhoven

Marver results

Finishing code took about 3 days, plus 5 day to fix the bugs that the user introduced.

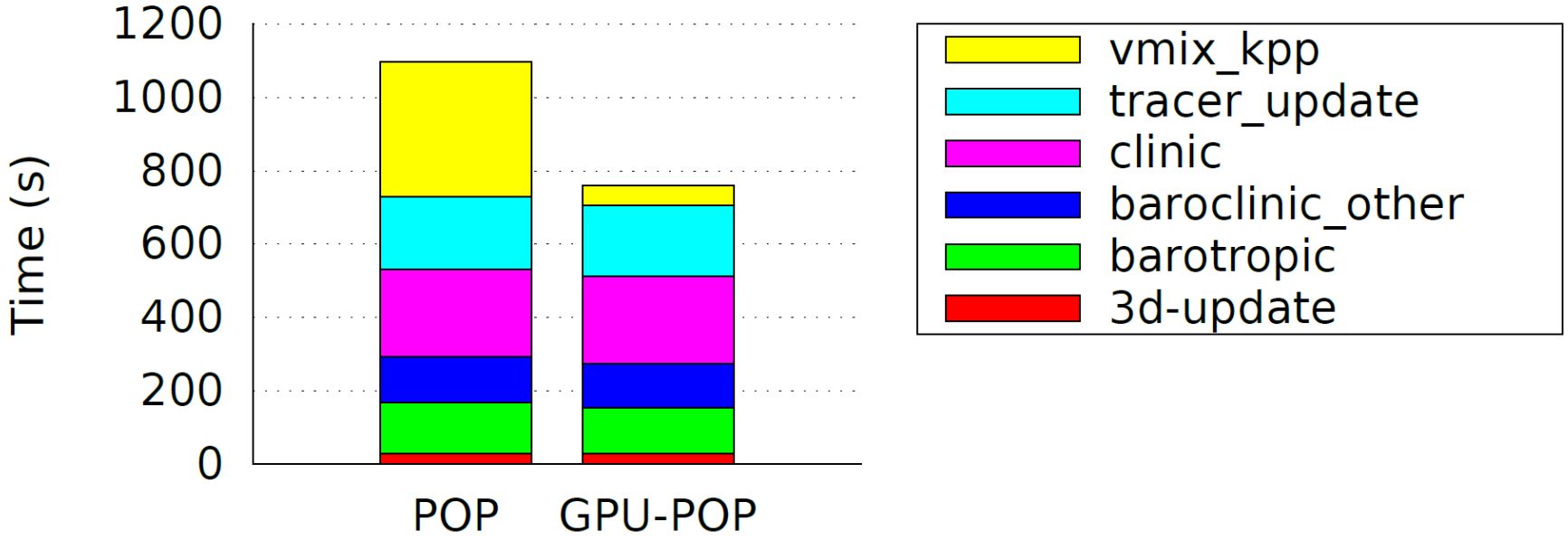
Porting effort is reduced significantly. It still requires good GPU skills, but saves a lot of the boring work!

original name	input	output	split	final
vmix_coeffs_kpp	152	232	vmix_coeffs_kpp_gpu_entry	299
buoydiff	102	84	buoydiff_kernel	71
			compute_vshearu	29
ri_iwmix	305	191	ri_iwmix_kernel	177
ddmix	129	100	ddmix_kernel	131
			compute_ustaru	15
			ugrid_to_tgrid	47
			vshearu_bldepth	21
bldepth	596	325	bldepth_kernel	297
			stability_and_buoyancy	34
blmix	434	279	blmix_kernel	280
interior_convection	231	143	interior_convection	80
			interior_part2	85
wscale	108	94	wsscale	29
			wmscale	29
smooth_hblt	182	100	smooth_hblt	100
ugrid_to_tgrid	58	41	ugrid_to_tgrid	40
tgrid_to_ugrid	58	41	tgrid_to_ugrid	29
sw_absorb_frac	75	61	sw_absorb_frac	53
state	410	131	mwjf_numerator	3
			mwjf_denominator	6
			compute_drhodt	12
			compute_drhods	9
Total	2840	1822		1876

Lines of code in the input (Fortran 90), output (CUDA/C), and final code

Source: Ben van Werkhoven

Marver results



Running POP on 8 GPU nodes of Cartesius (16 CPU cores, 2 K40 GPUs) shows a significant speedup in the yellow ported part

Source: Ben van Werkhoven

distributed computing

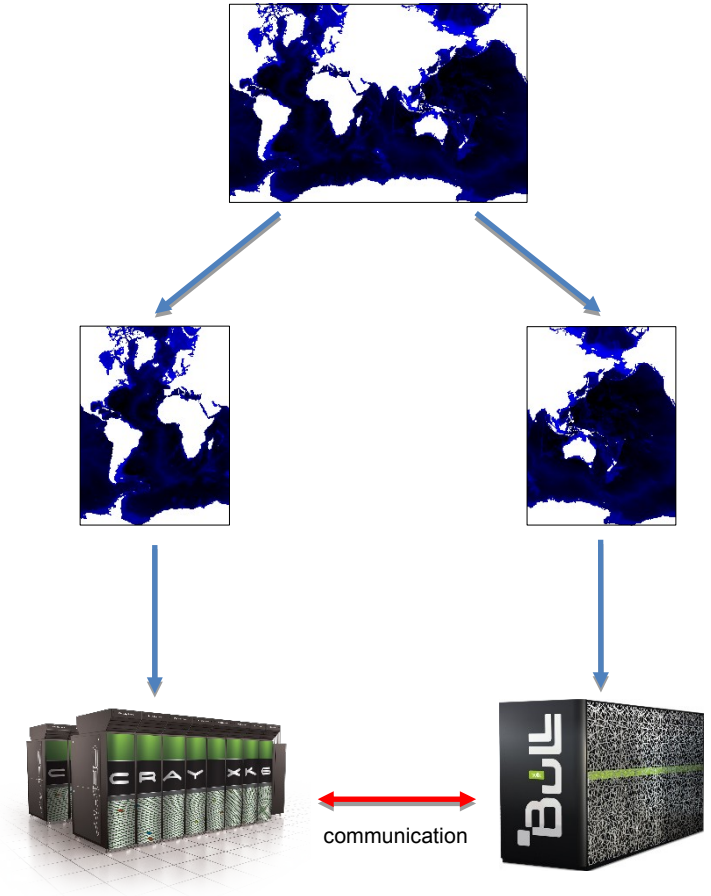
(the hard way)

Distributed Computing

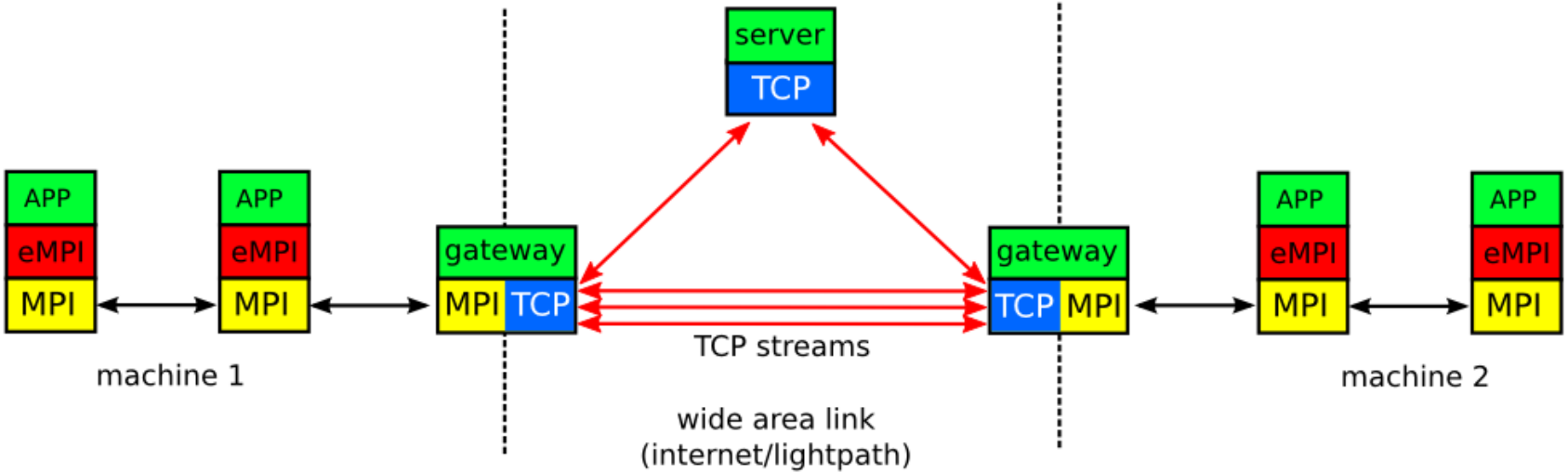
By combining supercomputers we can double / triple / … the amount of available compute power.

Example: split POP into 2 pieces and run each piece on a different supercomputer.

However: these supercomputers need to communicate (a lot) during the model run!



eSalsa-MPI connects supercomputers



eSalsa-MPI is an MPI wrapper library that can combine two MPI runs on different supercomputers, and presents it as a single set of processes to the application.

No change to the application is needed!

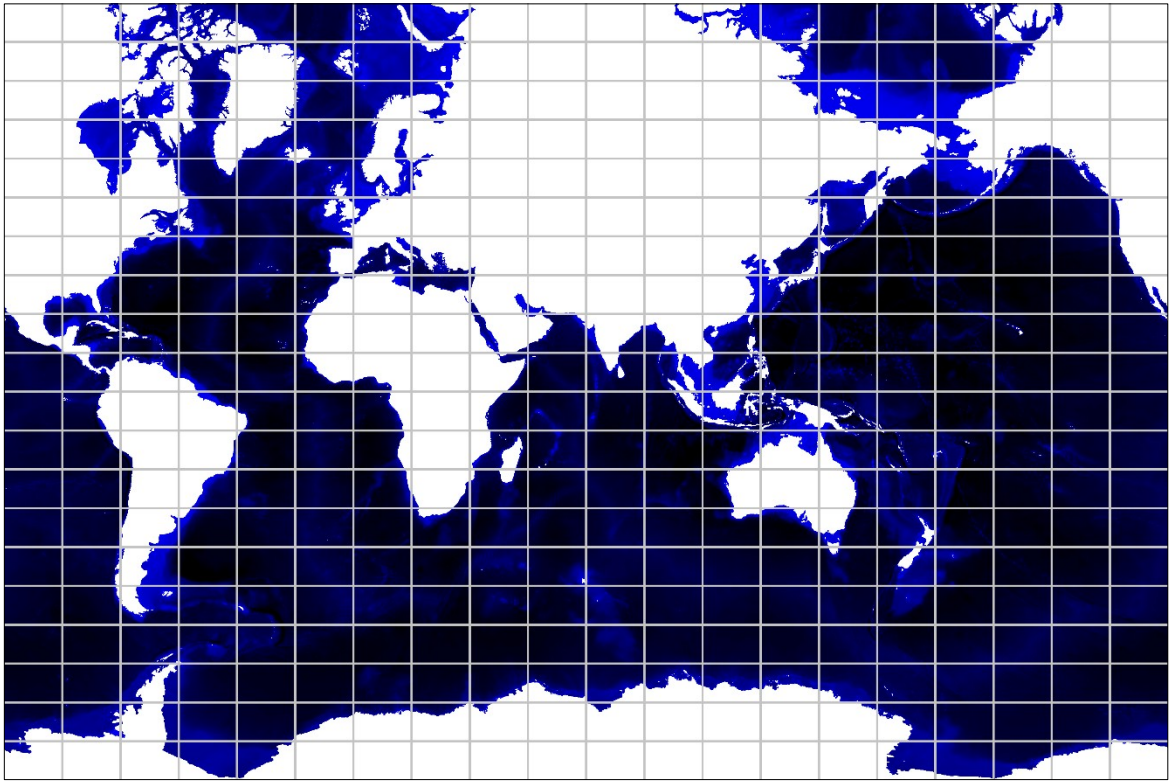
How to split POP ?

Remember:

Within one supercomputer POP also divides the world into blocks.

But there is a lot of communication between neighboring blocks!

How do you split the world in such a way that communication is minimized ?

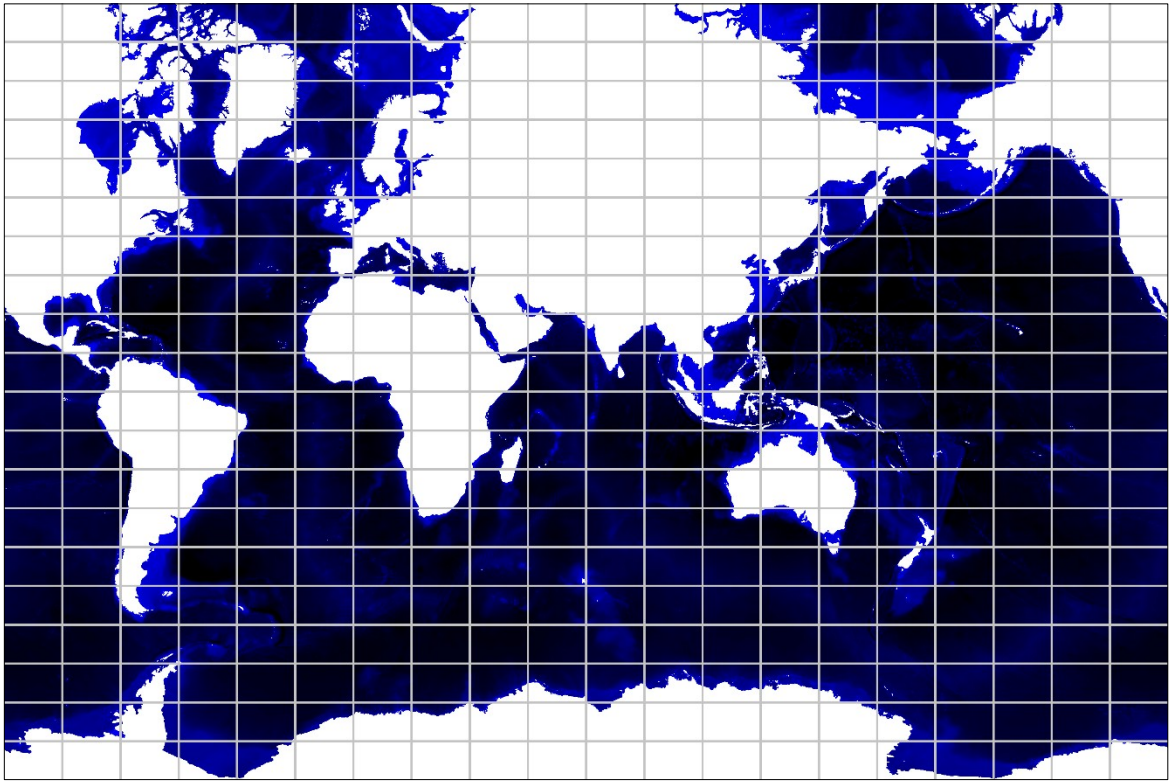


Discarding land-only blocks

Land-only blocks may be discarded.

Mixed ocean + land blocks are fully computed.

Example: 20x20 blocks (400) of 180x120x42 grid points each.



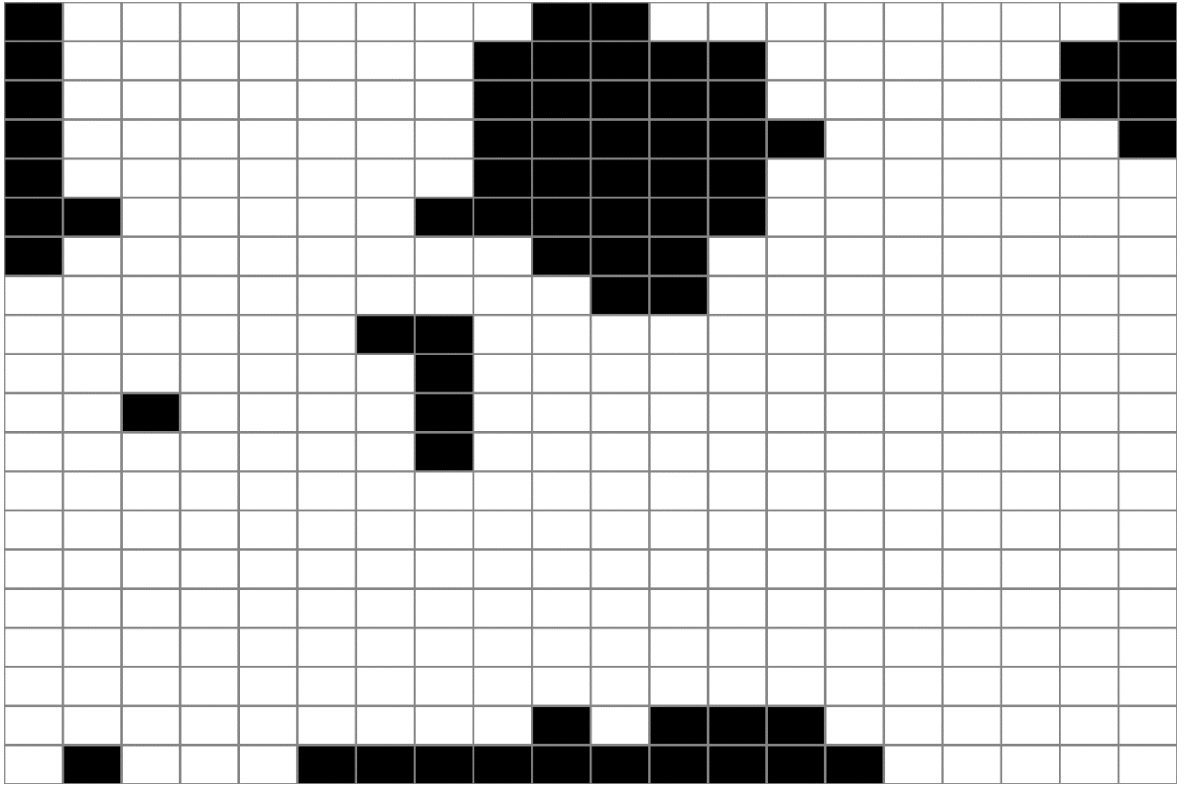
Discarding land-only blocks

Land-only blocks may be discarded.

Mixed ocean + land blocks are fully computed.

Example: 20x20 blocks (400) of 180x120x42 grid points each

Only 331 blocks are computed!



Using smaller blocks

To reduce the amount of useless computation, we can split the world into smaller blocks, and assign multiple blocks to each processor.

This is hard as it leads to load inbalance and extra communication overhead!

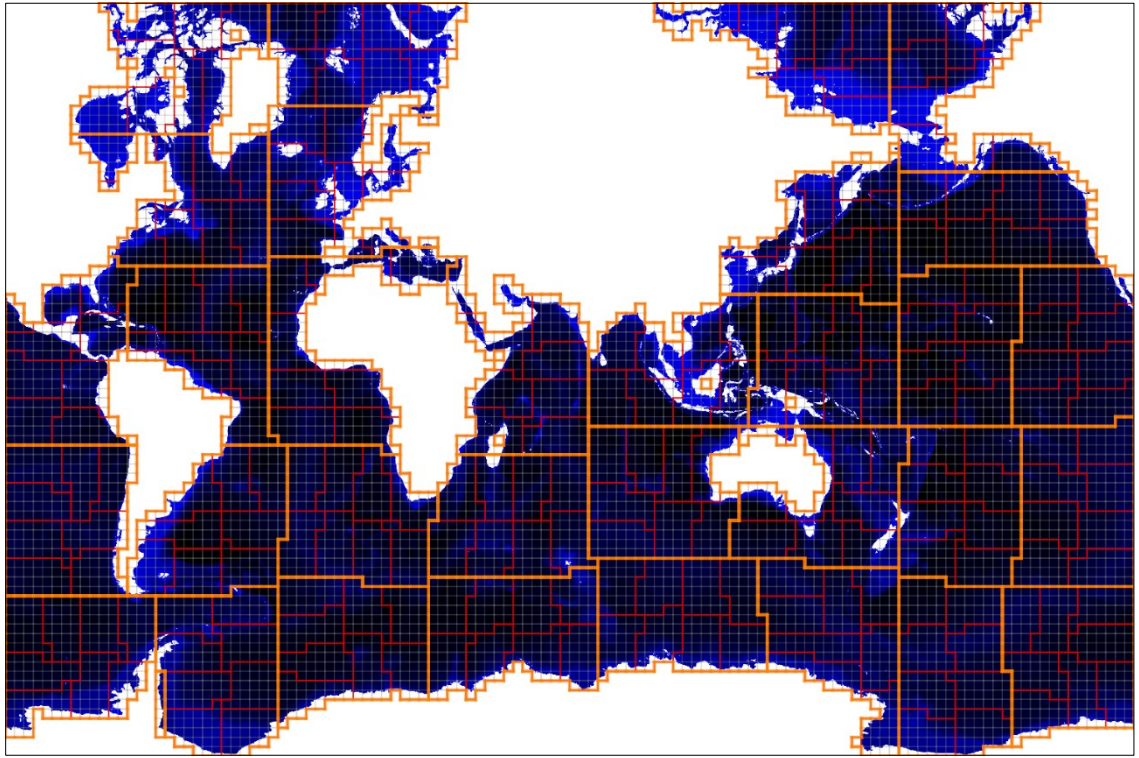
Hierarchical work distribution

To solve this problem we designed a hierarchical work distribution algorithm that recursively splits the world into blocks (top-down).

Example:

Split each result into 32 (nodes)
Split each result into 16 (cores)

Maximize load balance and minimize communication at each level!

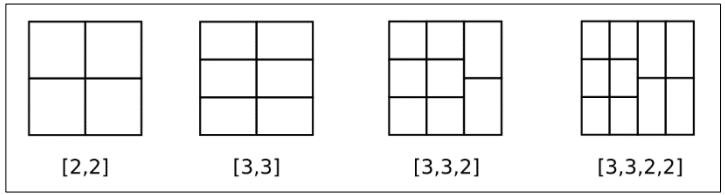


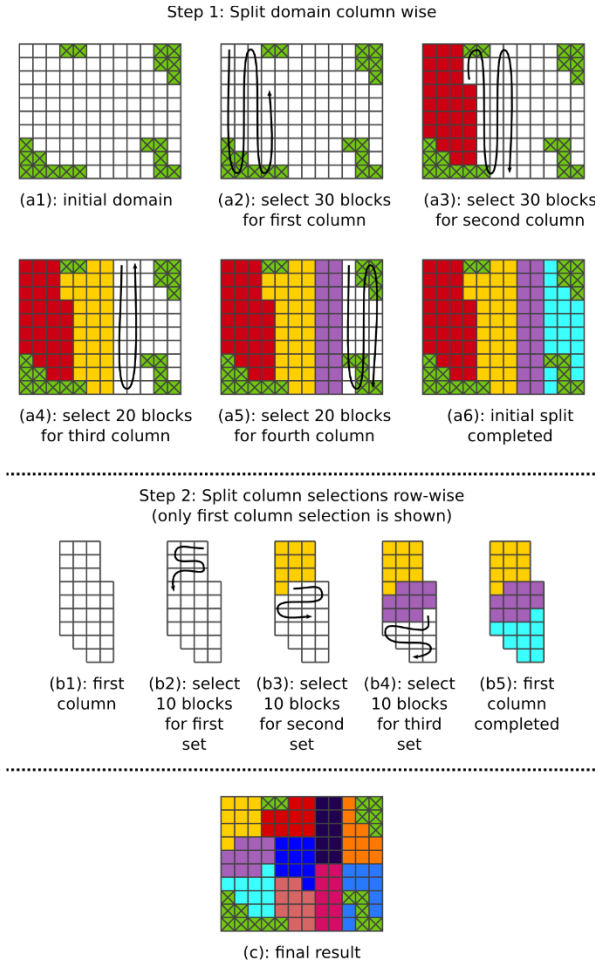
Source: A Distributed Approach to Improve the Performance of the Parallel Ocean Program
Ben van Werkhoven et al., Geoscientific Model Development, 7, 257-281, 2014

Algorithm

```
f := floor(sqrt(N));
c := ceiling(sqrt(N))

if (f = c)    We have found a square grid of [f x f]
if (f*c = N)  We have found a rectangular grid of [f x c]
if (N < f*c)  We have found a rectangular grid of [f x c] - (f*c-N)
if (N > f*c)  We have found a square grid of [c x c] - (c*c-N)
```

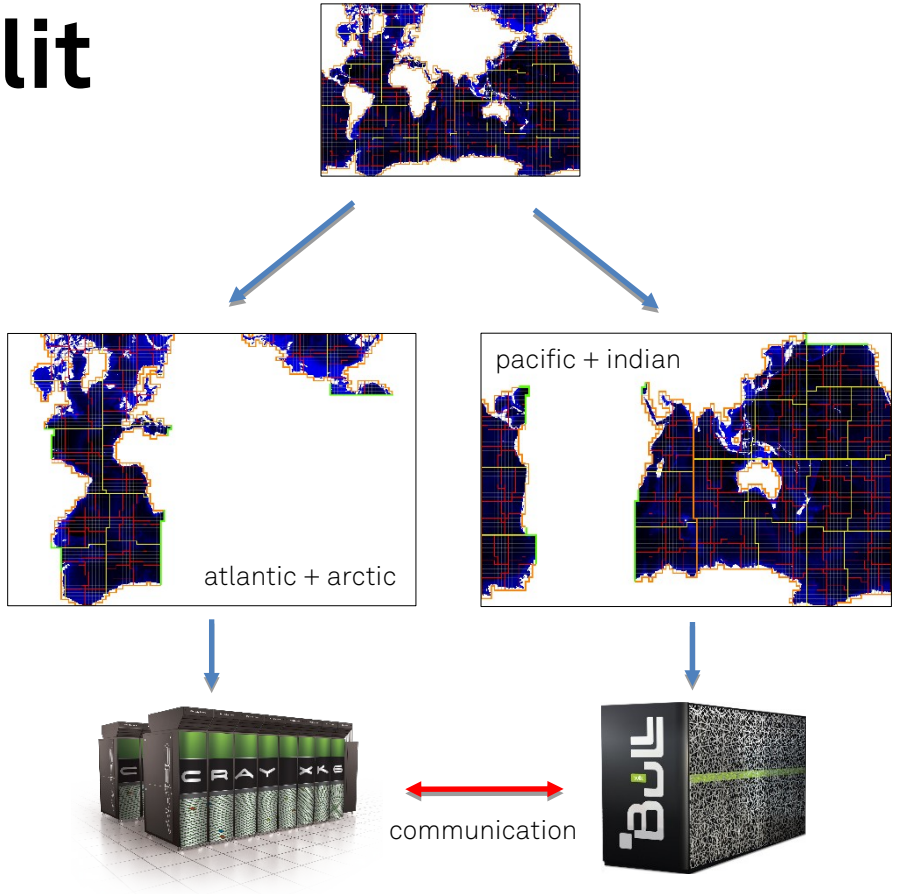




Geography aware split

We can easily extend the hierarchical work distribution scheme to include multiple supercomputers.

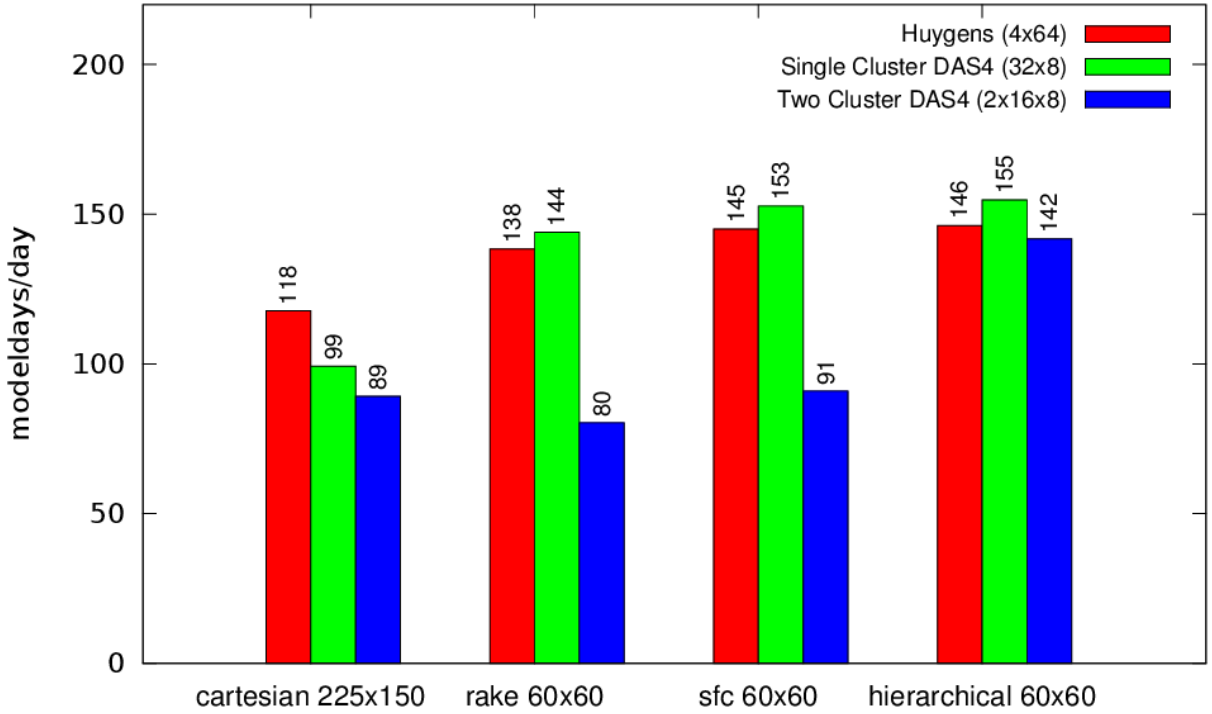
This results in a geography aware distribution of work, which significantly reduces communication between machines!



Performance of Distributed POP

Hierarchical work distribution performs well on two DAS4 cluster (256 cores) with only a shared 1 GBit/s link between them.

Produces 17% to 76% less communication between sites than other distributions.



Source: A Distributed Approach to Improve the Performance of the Parallel Ocean Program
Ben van Werkhoven et al., Geoscientific Model Development, 7, 257-281, 2014

Communication volume

Table 2. Configuration of the Cartesian, rake, and space-filling curve, and hierarchical distributions.

algorithm	block size	blocks per core (min/max)	blocks discarded	communication per task (min/avg/max)	communication between clusters (messages/volume)
Cartesian	225 × 150	1/1	0 (of 256)	0/1267.4/2408	22.3 M/99.0 GB
rake	60 × 60	5/8	628 (of 2400)	748/1940.5/3936	77.9 M/337.4 GB
space-filling curve	60 × 60	6/7	628 (of 2400)	1007/1707.7/2960	41.0 M/212.7 GB
hierarchical	60 × 60	6/7	628 (of 2400)	504/1394.9/2584	20.0 M/82.5 GB

Hierarchical work distribution reduces both message count and message volume between supercomputers.

Visualization

We visualize the output of the simulations to get a quick insight into the results.

Presenting eSalsa to King Willem-Alexander

distributed computing

(the next step)

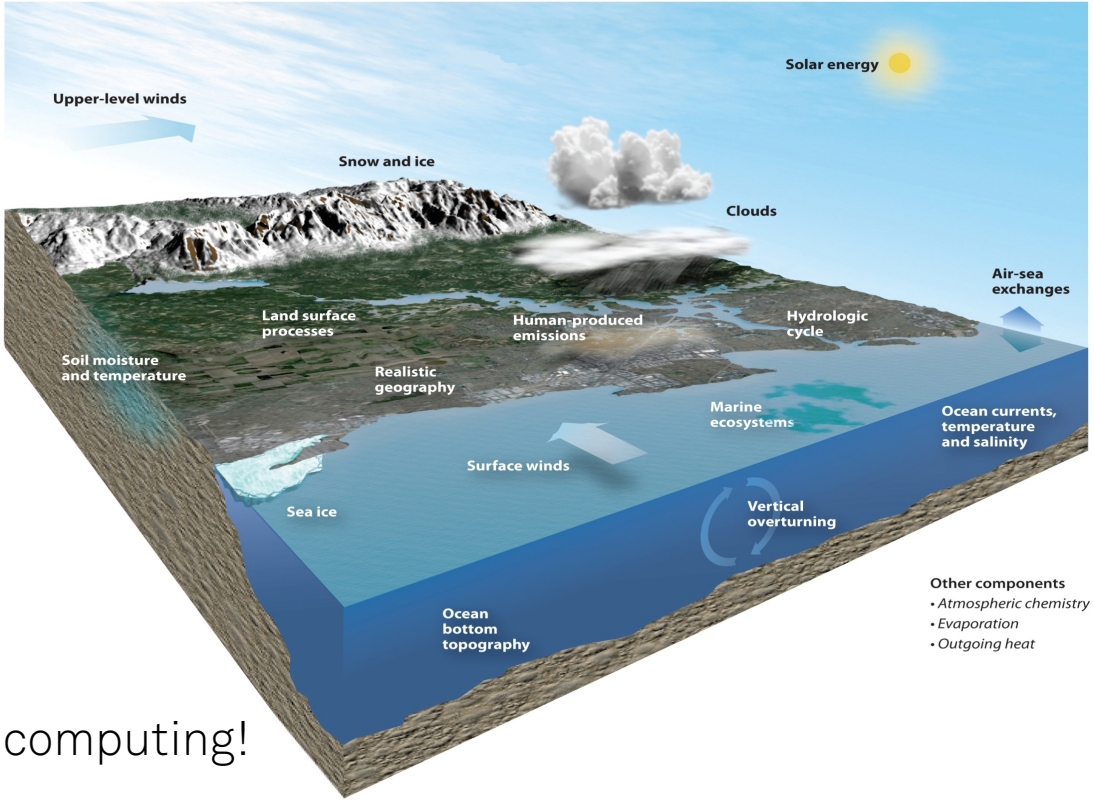
Earth System Modeling

Climate is more than just ocean!

We also want to combine atmosphere, ocean, land and ice models to simulate the interactions.

2~3x increase in compute time.

More opportunities for distributed computing!

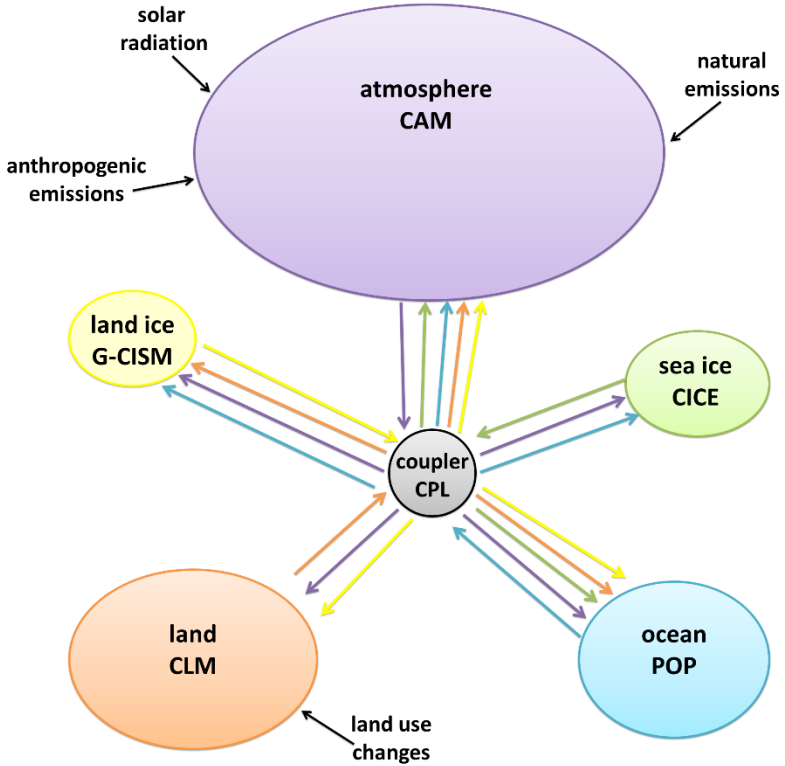


Community Earth System Model

CESM combines 5 simulations each simulating a different aspect of the climate (ocean, atmosphere, land, etc).

~1 million lines of Fortran / MPI

Models regularly exchange data to model the influence on each other.

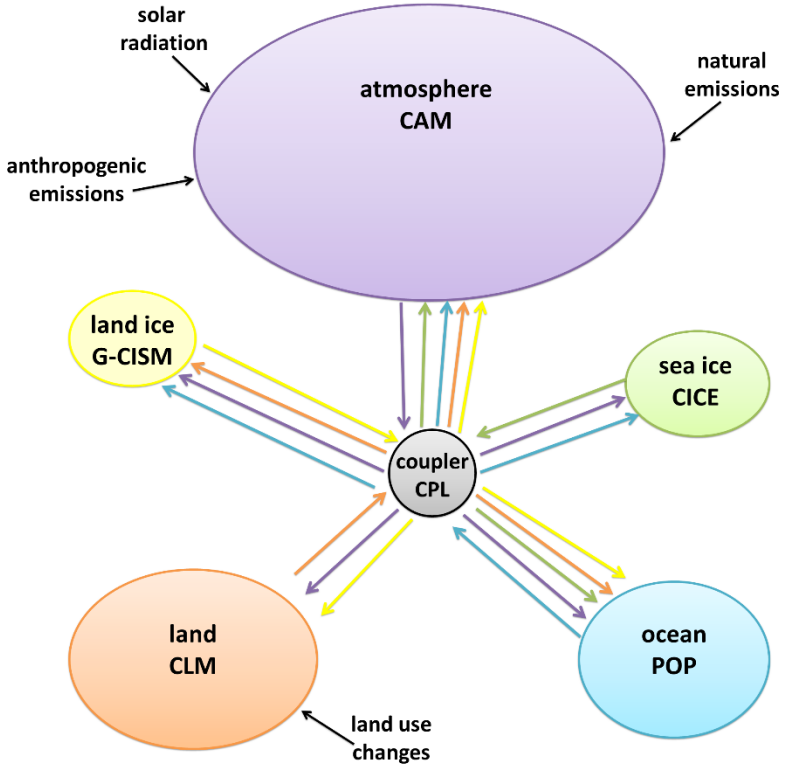


Source: climatesight.org

Community Earth System Model

Using distributed computing we run different submodels on different supercomputers.

This allows us to scale each submodel to higher resolutions and combine different architectures (for example a traditional machine + a GPU machine).

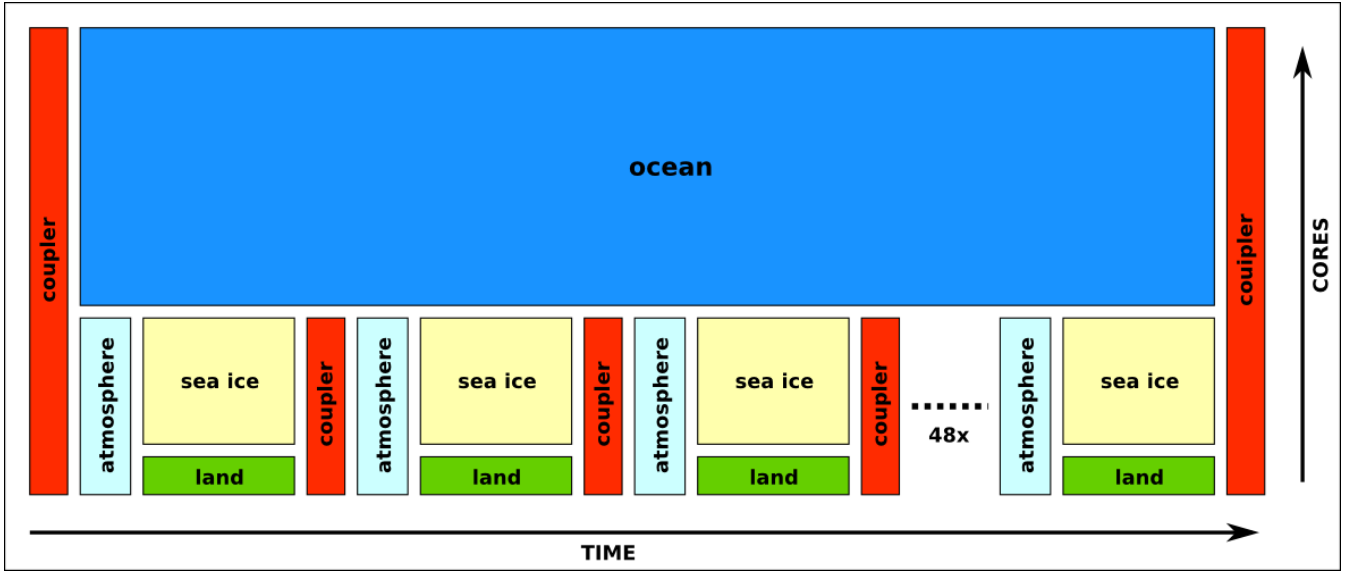


Source: climatesight.org

CESM model coupling

Data dependencies allow some models to run concurrently, while others must run in sequence.

(balancing their run times is black magic)

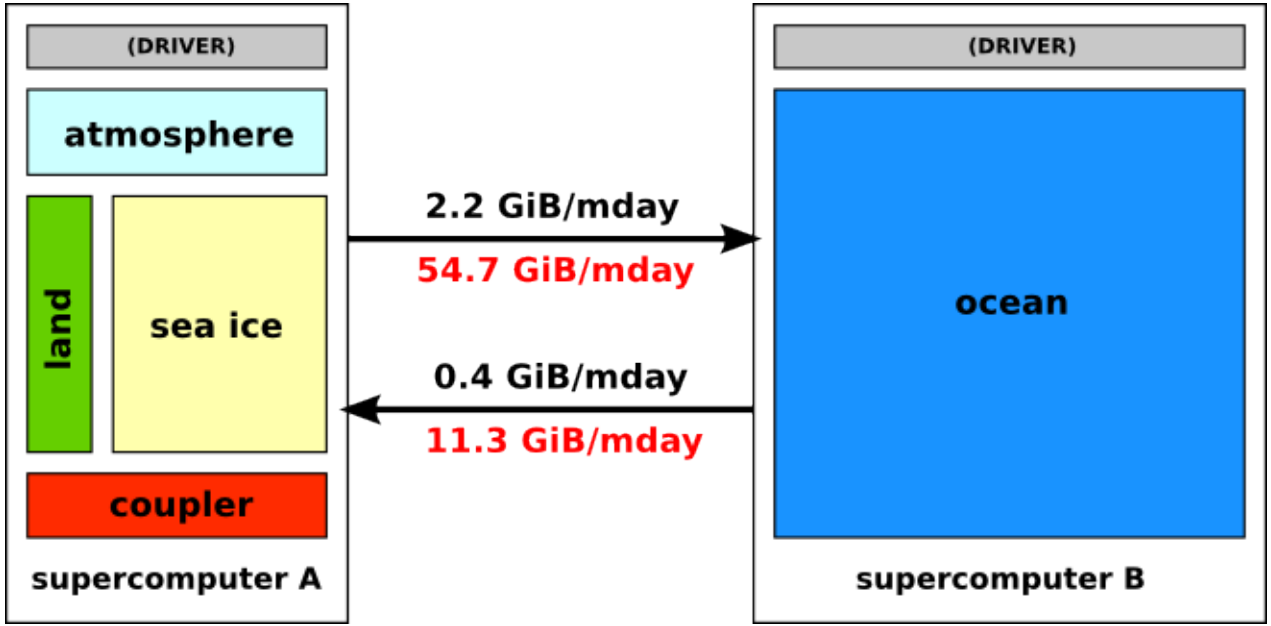


Atmosphere, land and sea ice exchange data every 30 model minutes, but ocean only 1x to 4x each model day!

Can we distribute CESM ?

We can run the ocean model separately

Coupling occurs 1~4x per simulated day (once every 30~120 sec) and data volume is limited.

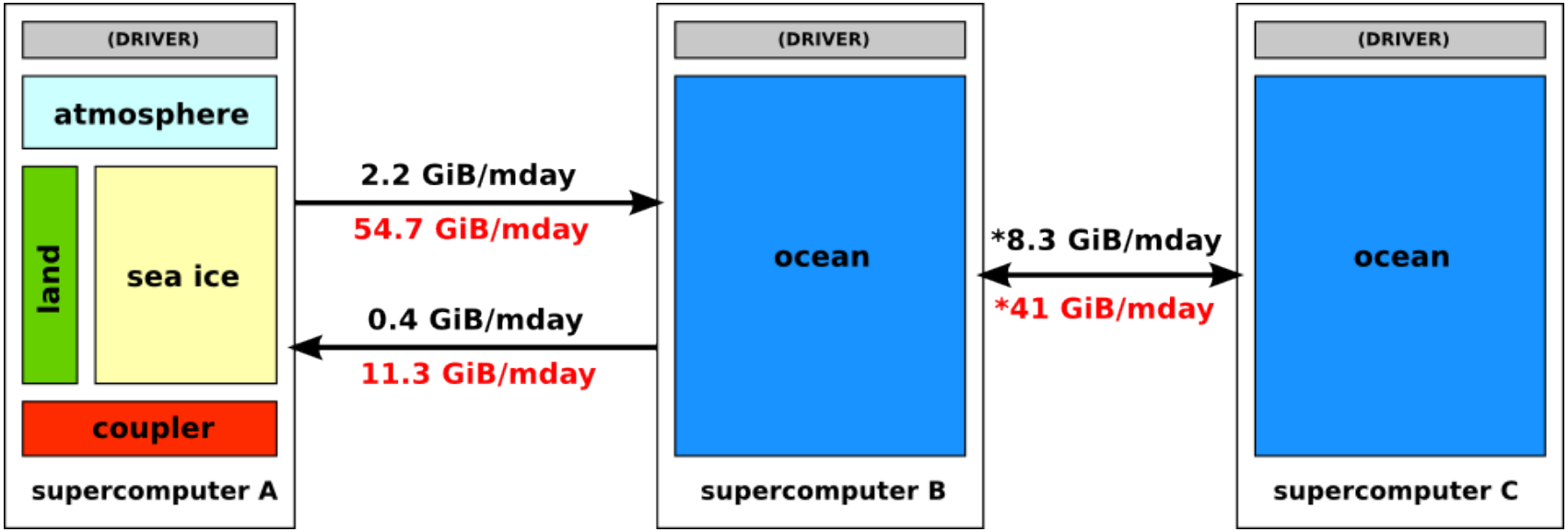


2.6 Gigabyte per exchange for 10 km resolution

66 Gigabyte per exchange for 2 km resolution

bursty traffic, not latency sensitive!

Can distribute CESM even more ?



Split ocean model using geography aware load balancing (GMD paper)

8.3 Gigabyte per exchange for 10 km resolution

41 Gigabyte per exchange for 2 km resolution

streaming traffic, latency sensitive!

netherlands eScience center

ENLIGHTEN YOUR RESEARCH GLOBAL

Top 500 #23
2.9 PFlop/s

SUPERMUC (GER)

10G

Top 500 #69
1.0 PFlop/s

CARTESIUS (NLD)

10G

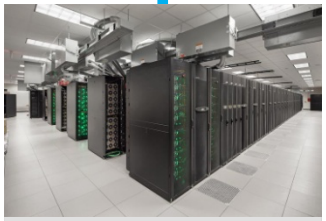
114 TFlop/s

EMERALD (UK)

GPUs

GPUs

10G



STAMPEDE (USA)

Top 500 #10
5.1 PFlop/s

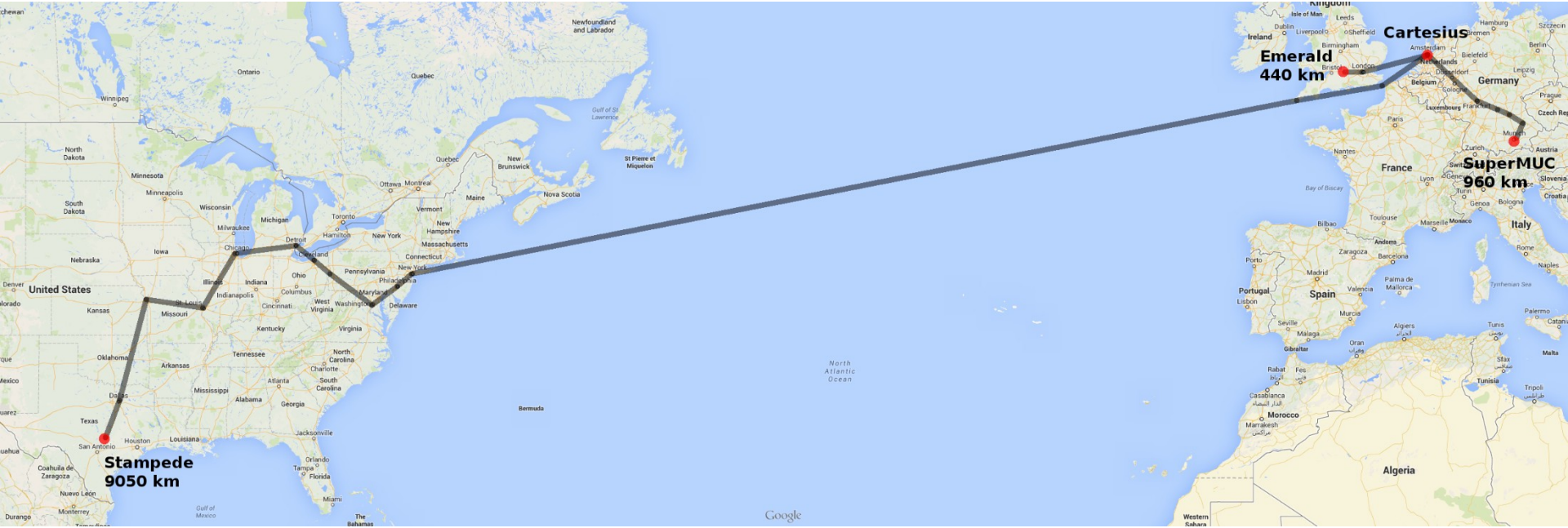
Xeon Phi

Idea: combine 2 or 3 supercomputers from EU and the US to run an Earth System Model at 0.02° (2x2 km) resolution.

(this required quite a bit lot of cooperation...)

IMAU Institute for Marine and Atmospheric research Utrecht
LMU Ludwig-Maximilians-Universität München
OXFORD e-Research CENTRE
UNIVERSITY OF OXFORD
NCAR National Center for Atmospheric Research
RUTGERS
RSMAS Rosenstiel School of Marine and Atmospheric Science, University of Miami
VU University Amsterdam
lrz
Universiteit Utrecht
netherlands eScience center
TACC
SURF SARA
netherlands eScience center

Our prize: World wide 10G network links!



NRENs provide the essential 10G network links between the sites.

netherlands eScience center

Supers are difficult to connect!

Arranging the infrastructure was quite hard!

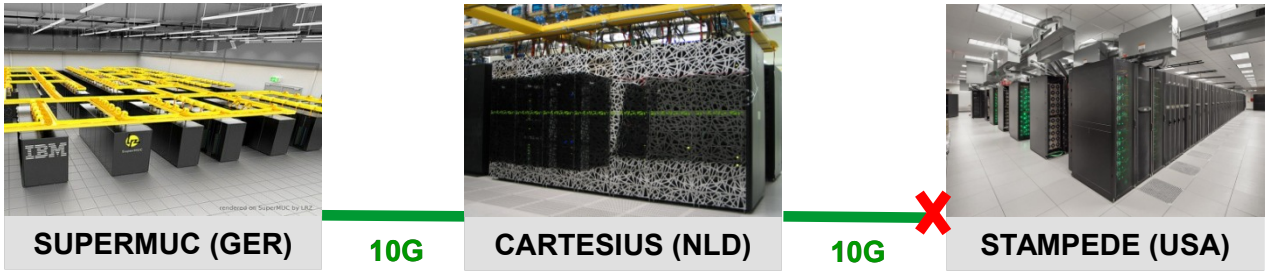
Interestingly, the software and the long distance links are not the hard part, but the endpoints are!

For eSalsa-MPI we need an extra machine connected to the wide area link and the internal network used by MPI – this was very hard to organize!

ENLIGHTEN YOUR RESEARCH GLOBAL

netherlands eScience center

Too much red tape….

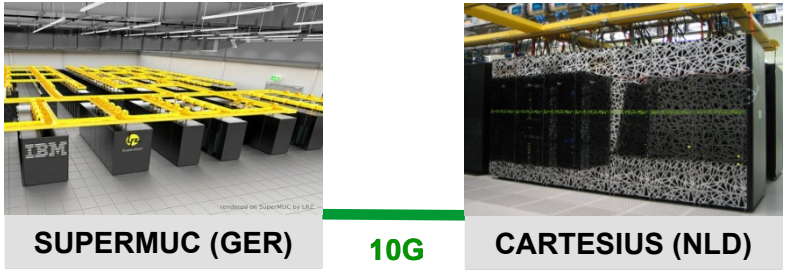


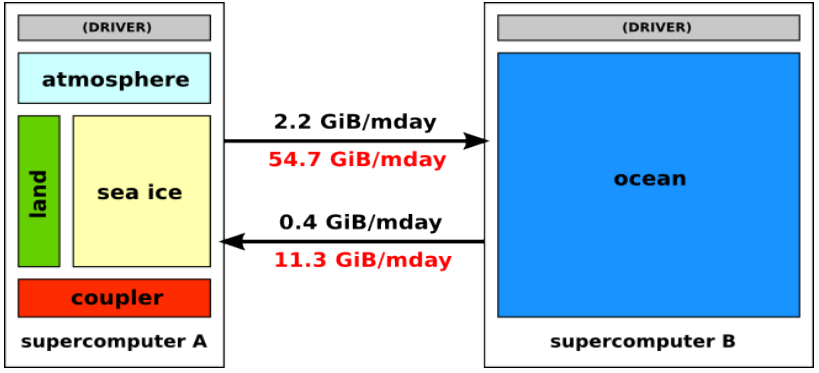
Supercomputers are usually not connected in this way, so we needed to cut through quite a bit of red tape. In the end we only managed to get the link to Germany up.

Initial results

Initial experiment are running on
912 cores at SuperMUC plus
608 cores at Cartesius (1520 total)

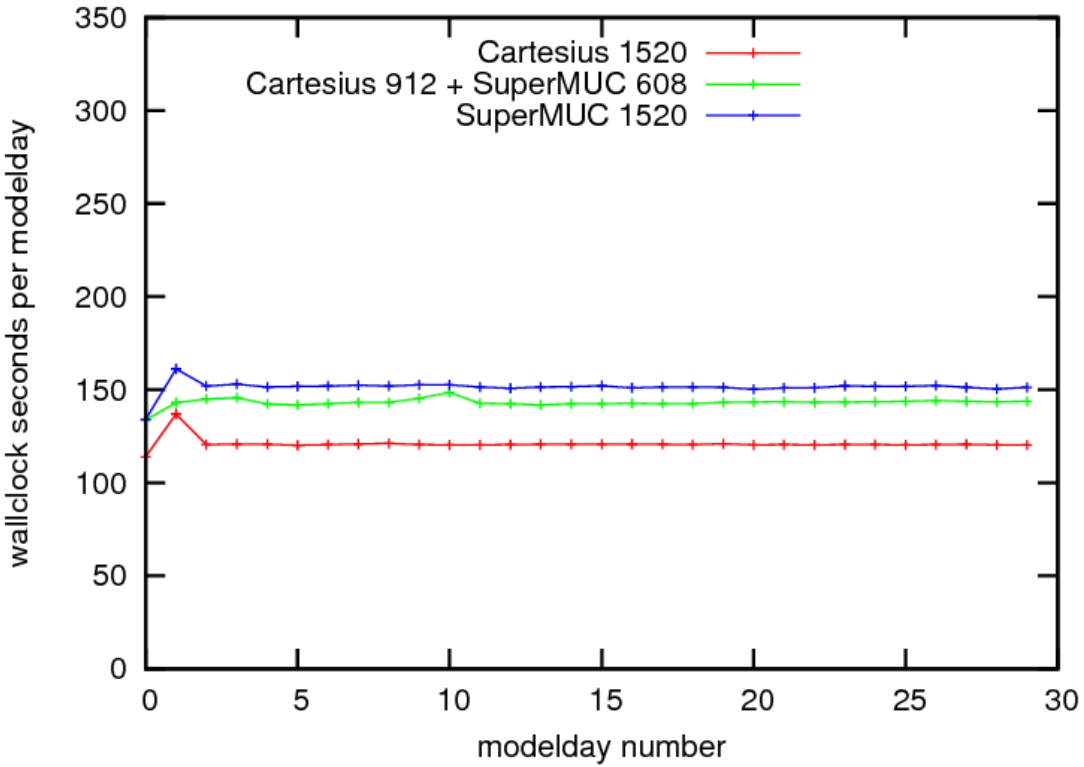
These weird core counts are determined by the limitations of the models, and their relative performance on both machines.





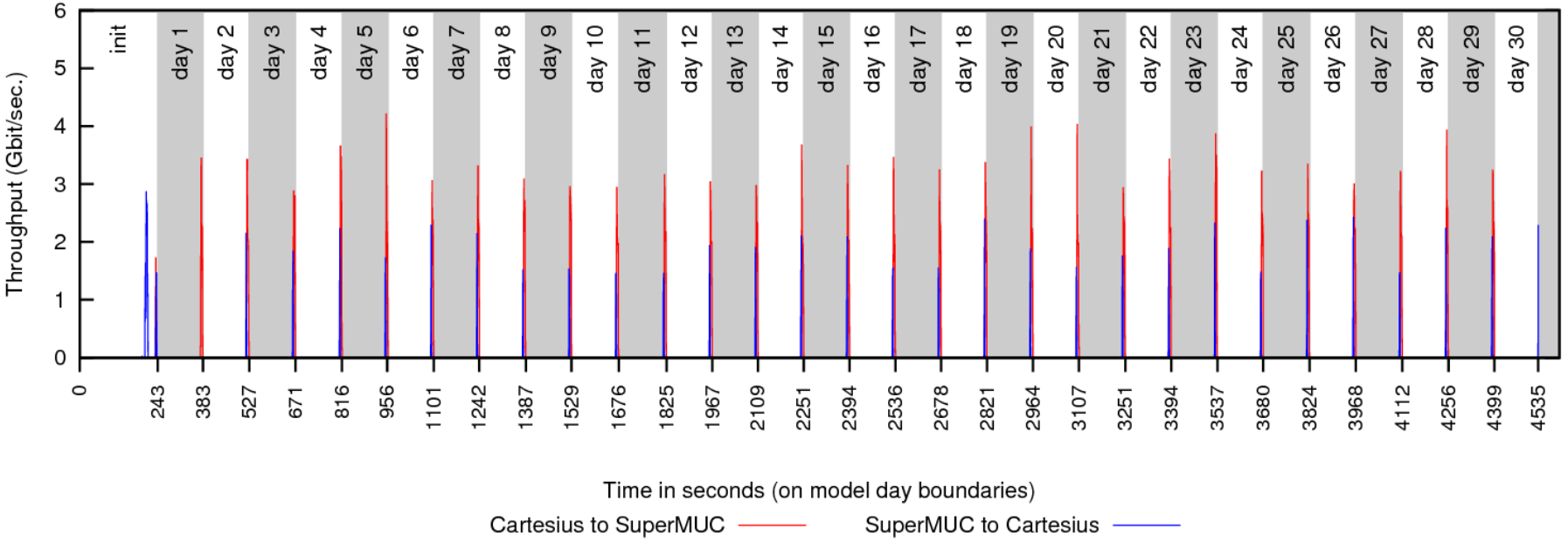
Results

Performance of combined system is “in between” the the individual systems at the same core count!



Wide area bandwidth

Cartesius - SuperMUC



Wide area communication on model day boundaries reaches 3~4 Gbit/s & 2 Gbit/s

eSalsa overview

In this project we used:

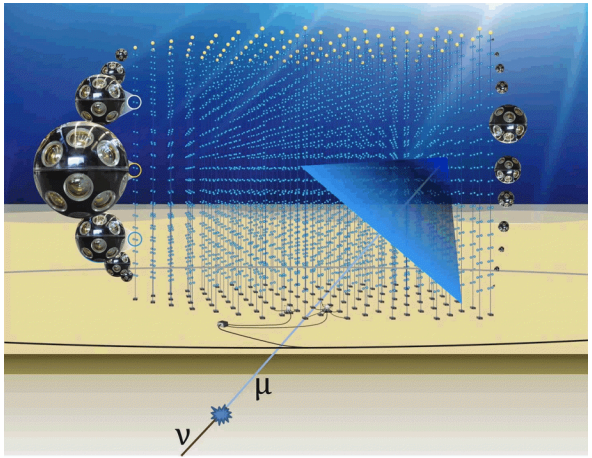
MPI, GPUs, 3 flavors of distributed computing, load balancing, optical wide area networks (lightpaths), performance models, semi-automatic translation from Fortran to CUDA, visualization of large data sets …

… we may have overdone it a little!

A few words
on
streaming sensor data

Keeping up with sensor data

LOFAR
25 GB/s



KM3NET
55 GB/s

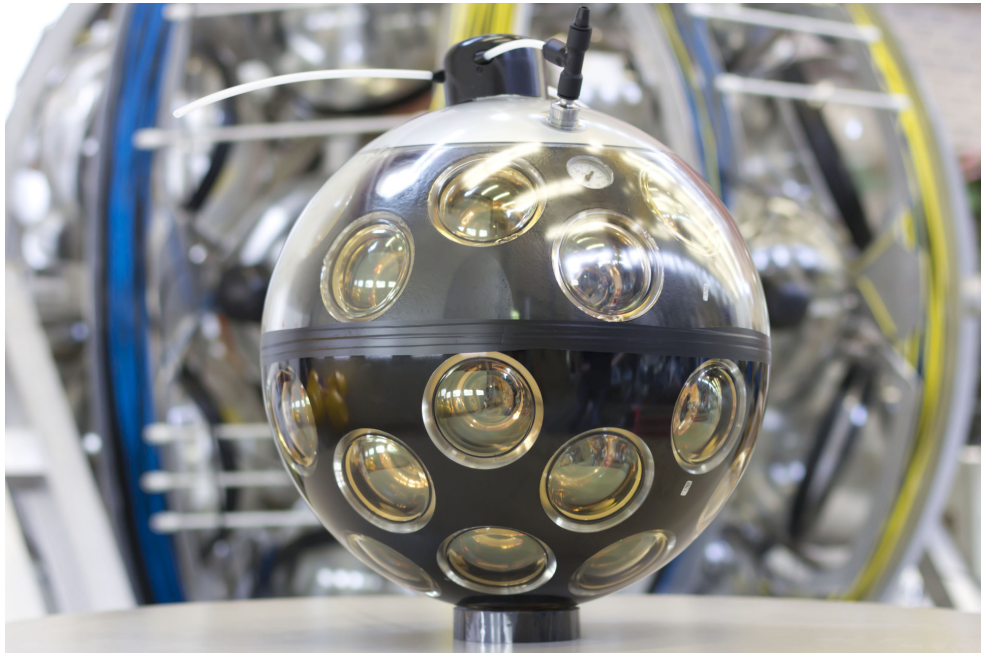
AMBER
7 GB/s

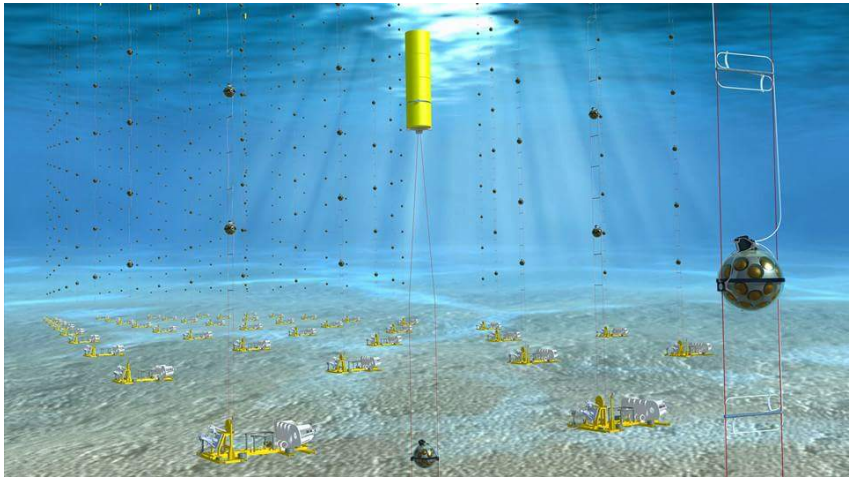
Many scientific instruments produce large amounts of data
that needs to be processed in real-time.

eAstronomy

LOFAR radio telescope – 88.000 antennas in 64 stations producing 25 GB/s, processed real time to remove radio frequency interference (cell phones, radio, lightning, etc), and to combine the signals into a single view of the sky.

Real-time detection of neutrinos from the distant Universe





KM3NeT neutrino detector:
12.000 spheres with 31 photo detectors on 600 strings in the Mediterranean. Process 55 GB/s in real time to remove interference from potassium decay and bio-luminescence. Produce output when a hit is likely (22 MB/s).

Real-time SAR processing

AMBER Synthetic Aperture Radar:
8 sending and 24 receiving antennas, whose signals are combined in software. Requires a quite a bit of processing, but on a limited energy budget of about 50 watts.

GPUs are essential

(of all shapes and sizes)

Only GPUs deliver the raw performance **and** performance per watt needed for these projects.
The same expertise is required for the big telescopes and the small drones!

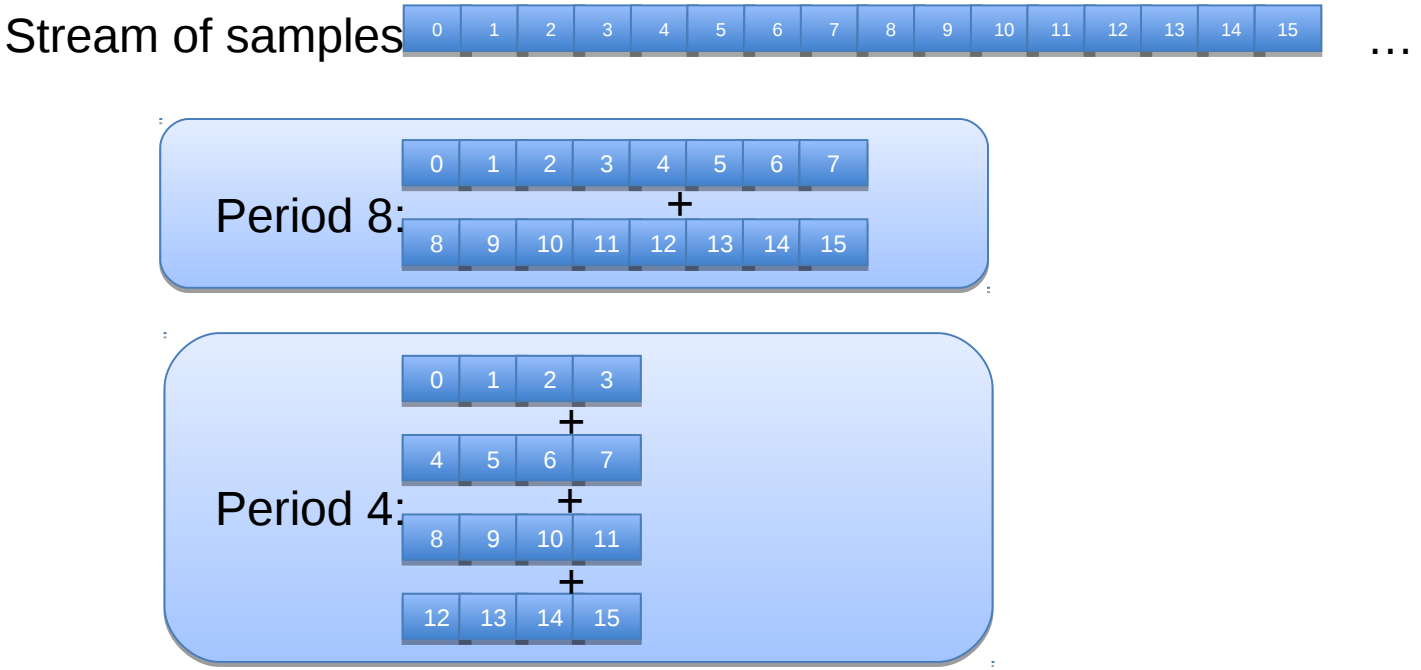
Straightfoward algorithms

(LOFAR correlator)

```
for (time = 0; time < integrationTime; time++) {
    sum += samples[ch][station1][time][pol1] *
               samples[ch][station2][time][pol2];
}
```

Straightfoward algorithms

(LOFAR RFI mitigation)



Straightfoward algorithms

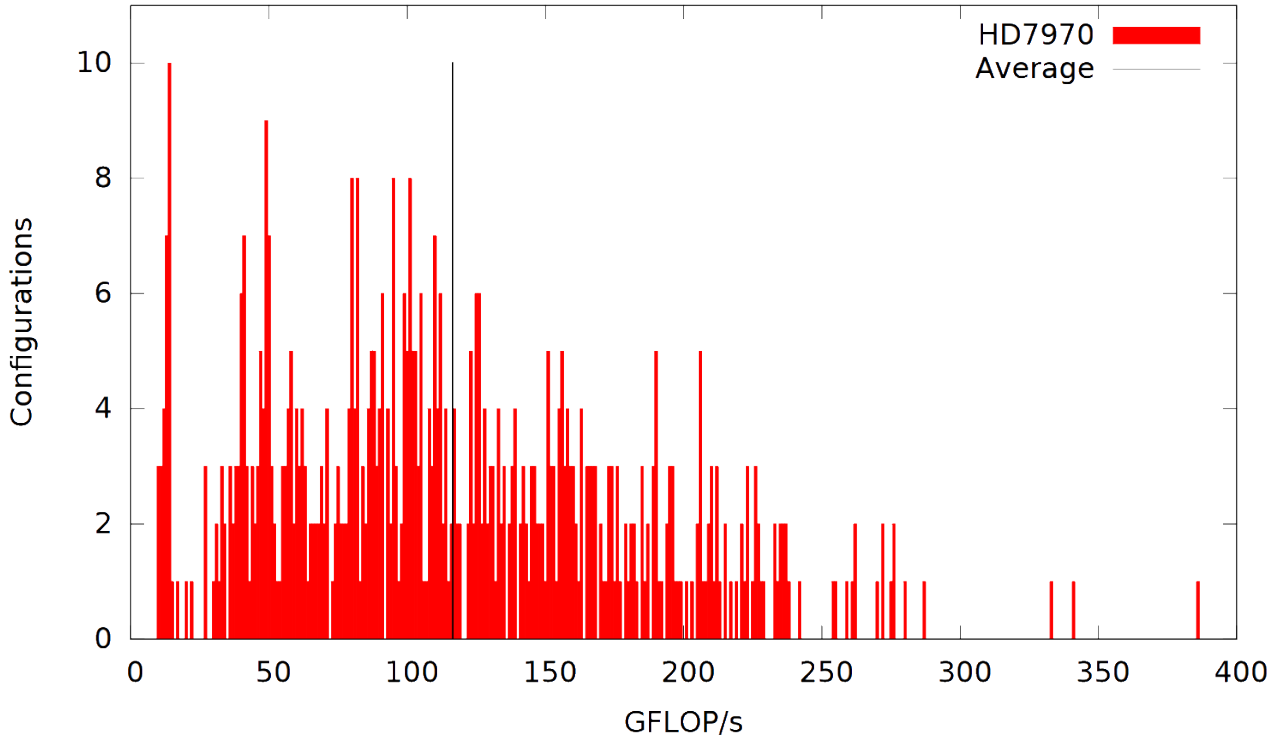
(KM3NeT hit correlation)

For every pair of detected hits (x_1, y_1, z_1, t_1) and (x_2, y_2, z_2, t_2) compute:

$$(t_1 - t_2)^2 \times c < (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2$$

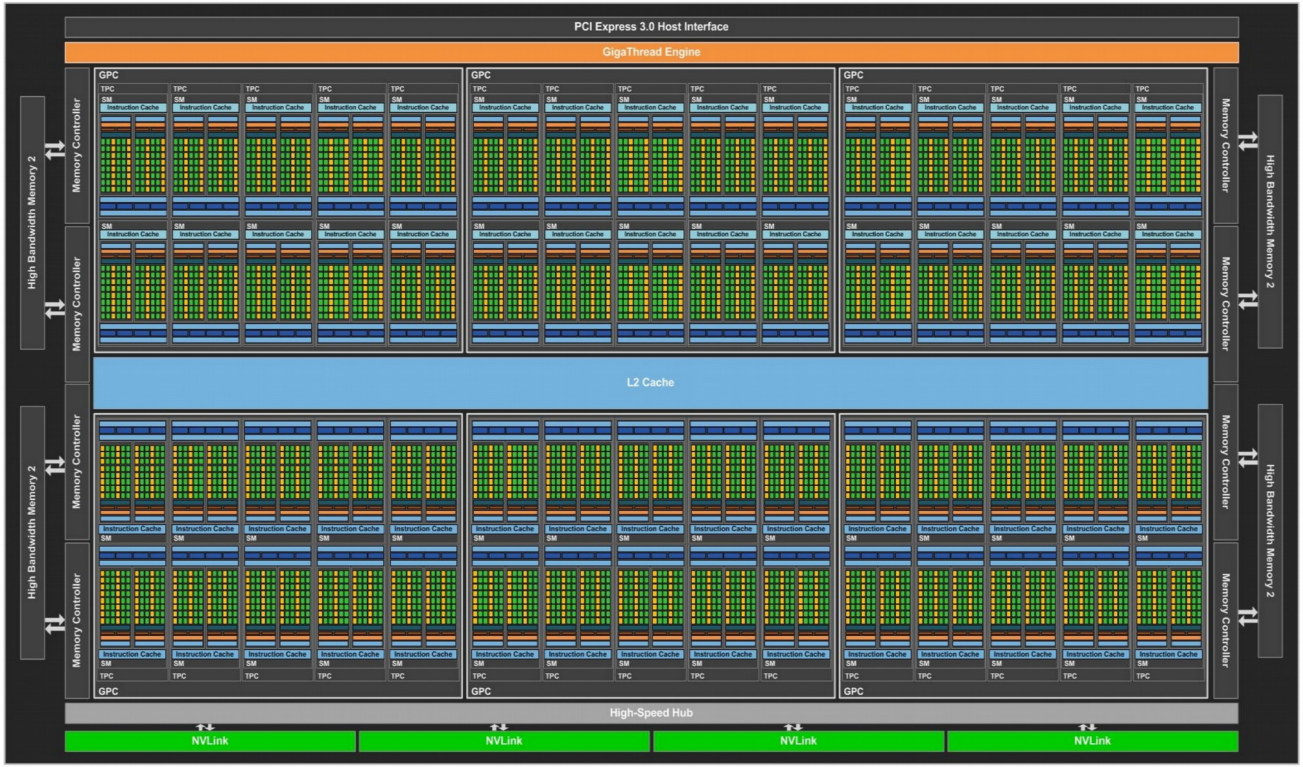
Tuning GPUs codes is a lot of work

(many possibilities)

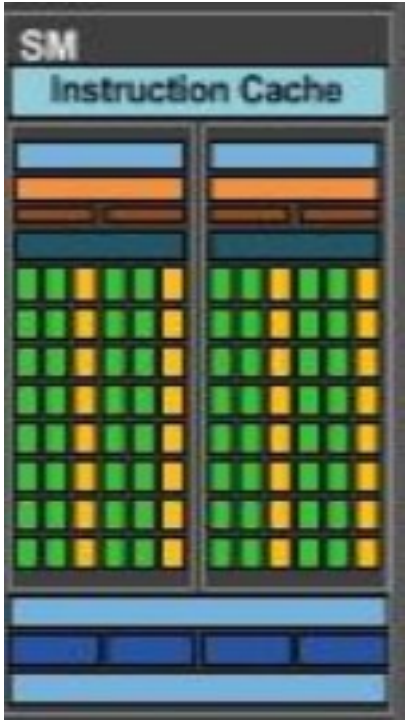


(*Real-Time Pulsars Pipeline Using Many-Cores.* Sclocco et al. 2014)

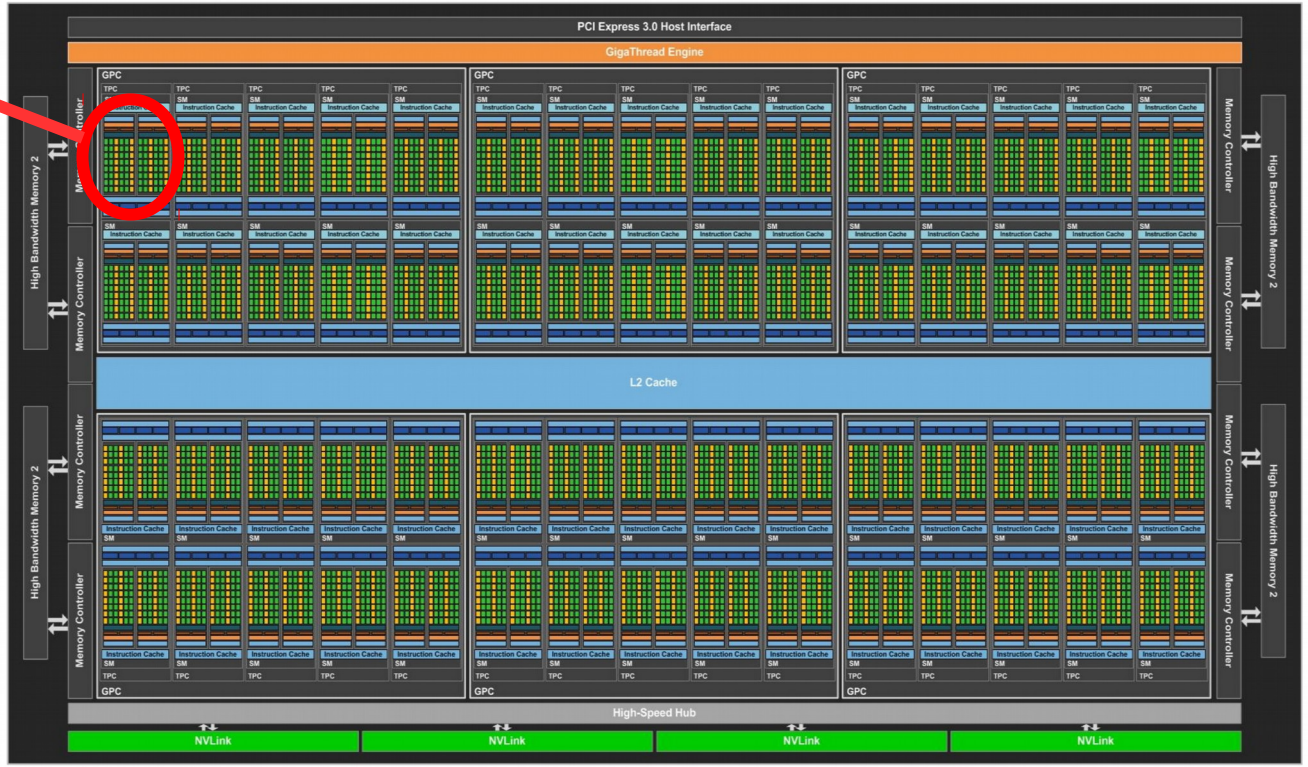
Why is this hard?



Massively parallel!



Why is this hard?



60 SMs
64 cores each
64KB shared memory

Matrix Multiply

GPU architecture is visible in the code:

Work is divided into blocks each containing a number of threads that can share memory:

- 2D block index:
  blockIdx.x, blockIdx.y, block_size

- 2D thread index (within 1 block):
  threadIdx.x, threadIdx.y

- Shared memory (within 1 block)
  __shared__

```
__global__ void matmul_kernel(float *C, float *A, float *B) {

    __shared__ float sA[block_size][block_size];
    __shared__ float sB[block_size][block_size];

    int tx = threadIdx.x;
    int ty = threadIdx.y;
    int x = blockIdx.x * block_size + tx;
    int y = blockIdx.y * block_size + ty;

    float sum = 0.0;
    int k,kb;

    for (k=0; k<WIDTH; k+=block_size) {
        __synchthreads();
        sA[ty][tx] = A[y*WIDTH+k+tx];
        sB[ty][tx] = B[(k+ty)*WIDTH+x];
        __synchthreads();

        for (kb=0; kb<block_size; kb++) {
            sum += sA[ty][kb] * sB[kb][tx];
        }

    }

    C[y*WIDTH+x] = sum;
}
```

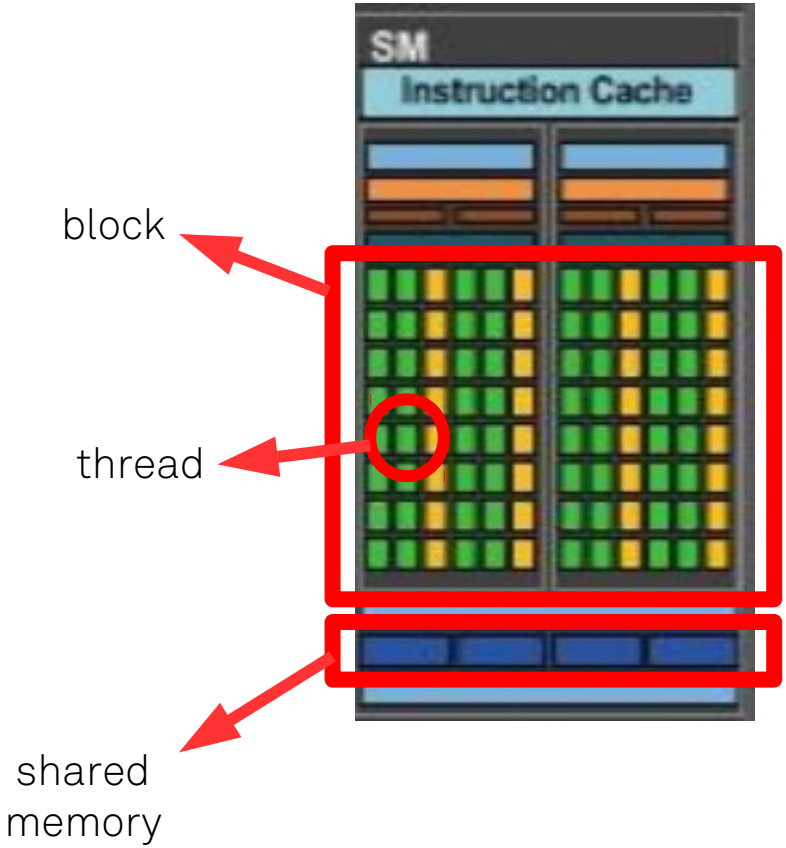
Matrix Multiply

It is very important to choose the best **block_size, thread dimesions, etc.**

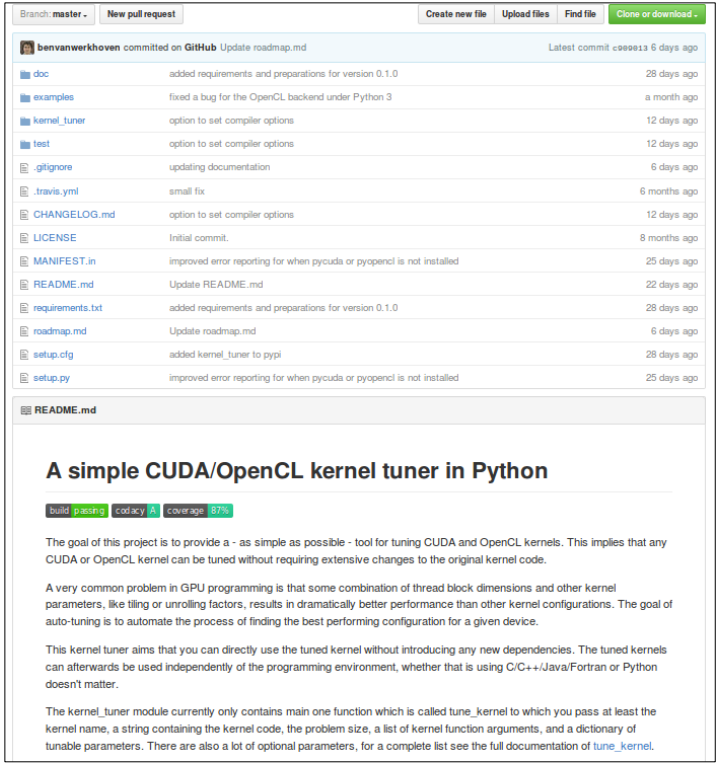
When chosen correctly, the algorithm **maps directly** onto the SMs, cores and shared memory of the GPU, resulting in good performance.

When chosen incorrectly the algorithm does not map nicely, and the performance **drops fast!**

Many algorithms have tens of such parameters!



Kernel Tuner



Optimizing the choice of these parameters can be partly automated.

The **kernel tuner** is a tool that automatically searches for the optimal configuration options for a piece of code provided by the user, thereby saving a lot of a programmers time.

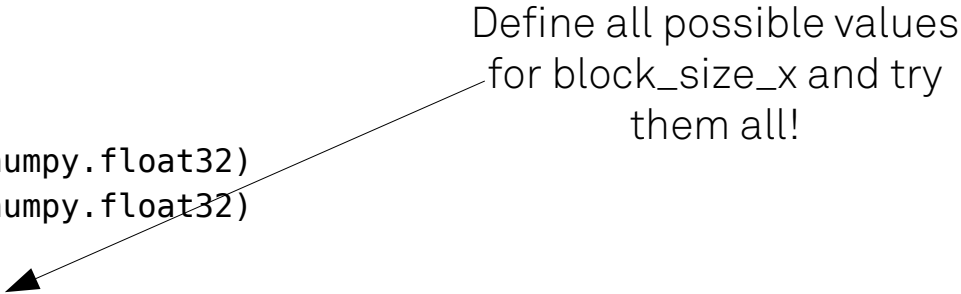
Porting the application to another machine is reduced to simply re-tuning it.

Kernel Tuner Example

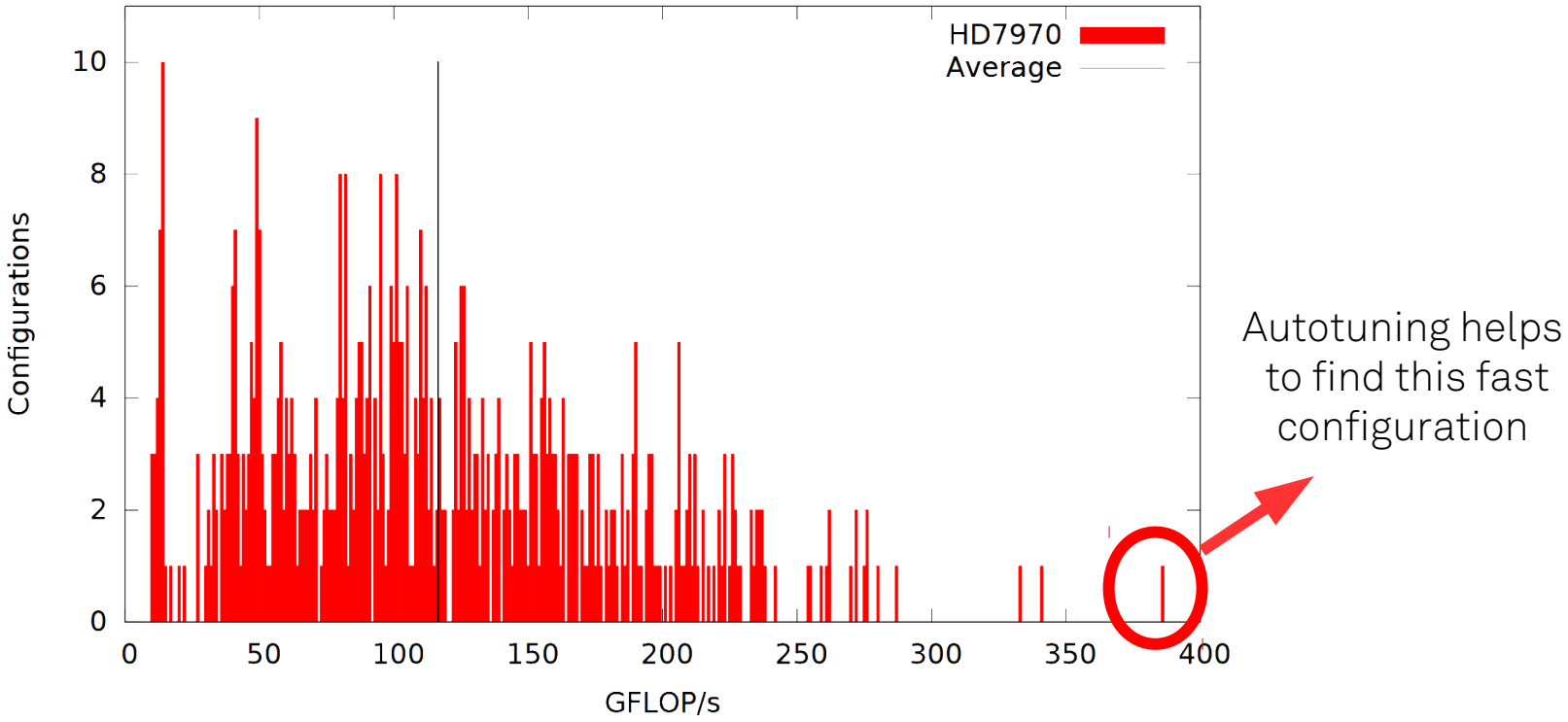
```
kernel_string = """
__global__ void vector_add(float *c, float *a, float *b, int n) {
    int i = blockIdx.x * block_size_x + threadIdx.x;
    if (i<n) {
        c[i] = a[i] + b[i];
    }
}"""

n = numpy.int32(1e7)
a = numpy.random.randn(n).astype(numpy.float32)
b = numpy.random.randn(n).astype(numpy.float32)
c = numpy.zeros_like(b)
args = [c, a, b, n]
tune_params = {"block_size_x" : [128+64*i for i in range(15)] }

kernel_tuner.tune_kernel("vector_add", kernel_string, (n,1), args, tune_params)
```



Tuning goal



(Real-Time Pulsars Pipeline Using Many-Cores. Sclocco et al. 2014)

Streaming Sensor data overview

Many sensors generate enormous amount of data and need real-time processing

GPUs are a good choice here, but GPU codes are hard to performance tune

Fortunately, this can be automated using tools like the **Kernel Tuner**

In conclusion ...

Real world applications often use a combination of parallelization types (MPI, OpenMP, CUDA, hierarchical simulations, ensembles, etc.)

Incorporating GPUs is often non-trivial, but potential gains are high.

Solutions are often application specific, but the knowledge and tools surrounding them are not (Xenon, Kernel Tuner, eSalsa-MPI, Marver, etc).

The goal of the eScience Center is to extract these generic knowledge and tools and make them available to all of Dutch academia.

Get in touch

Netherlands eScience Center
Science Park 140
1098 XG Amsterdam
The Netherlands

+31 (0)20 4604770
info@eScienceCenter.nl

www.eScienceCenter.nl

linkd.in/1j2uS8S

vimeo.com/eScienceCenter

twitter.com/eScienceCenter

j.maassen@esciencecenter.nl

(we also do MSc projects!)