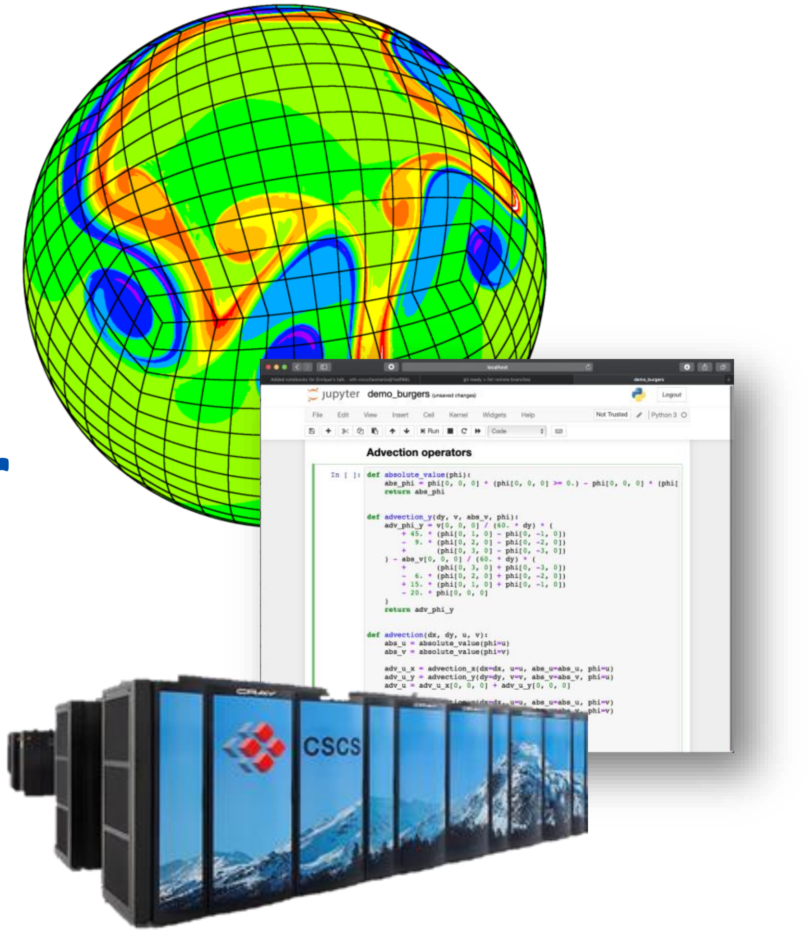
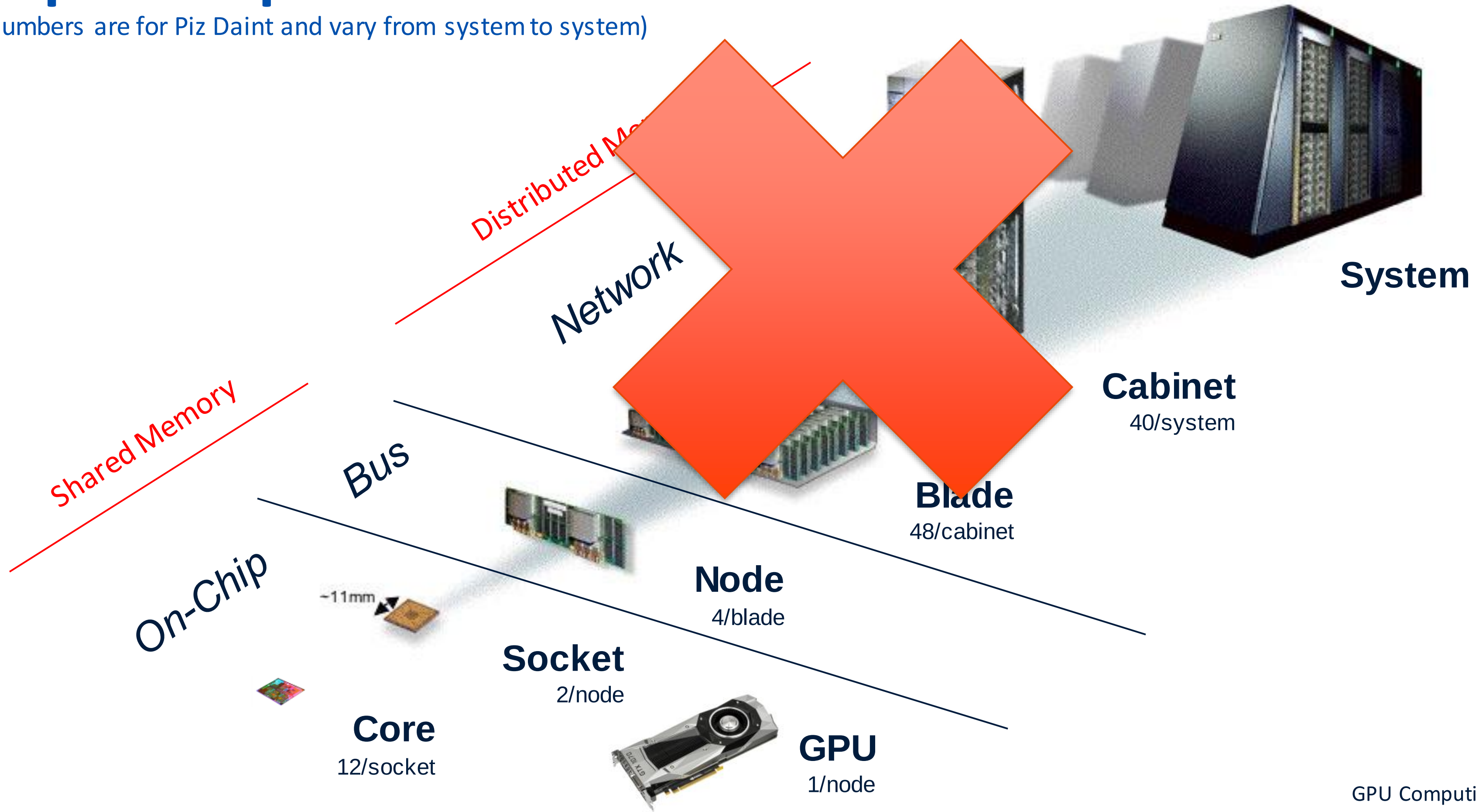
High Performance Computing for Weather and Climate (HPC4WC)

Content: Graphics Processing Units
Lecturer: Tobias Wicky
Block course 701-1270-00L
Summer 2023



Supercomputer Architecture

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



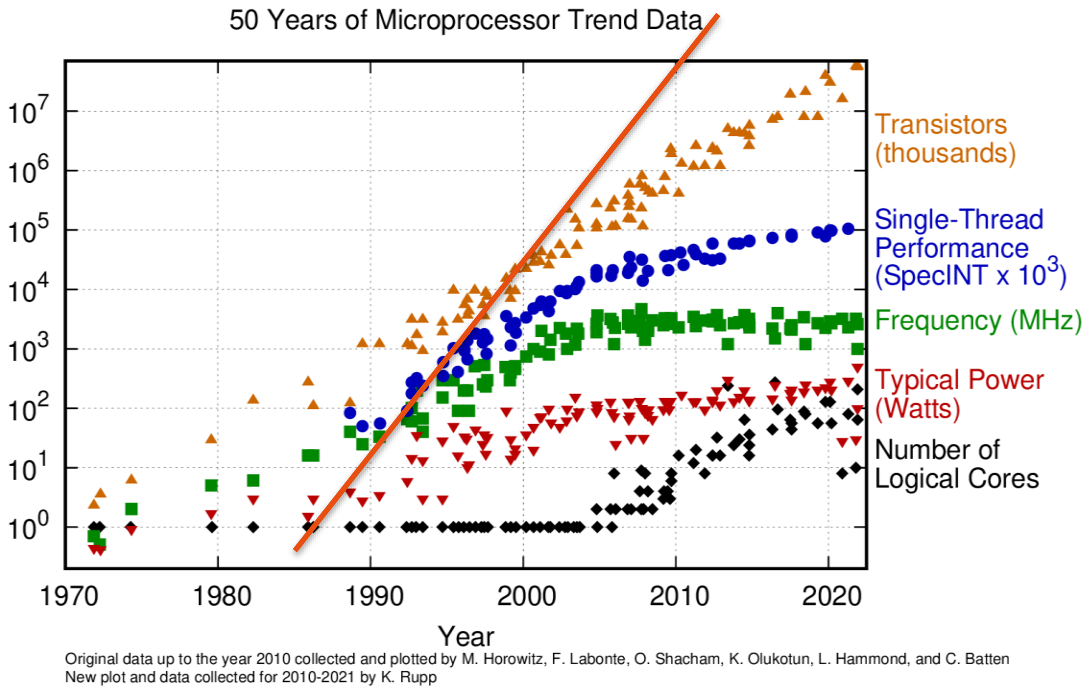
Learning goals

- Understand why specialized hardware such as GPUs is become the new "normal"
- Learn how to program a GPU using a high-level programming language
- Understand potential and difficulties of GPU-computing

How does the landscape of HPC look today?

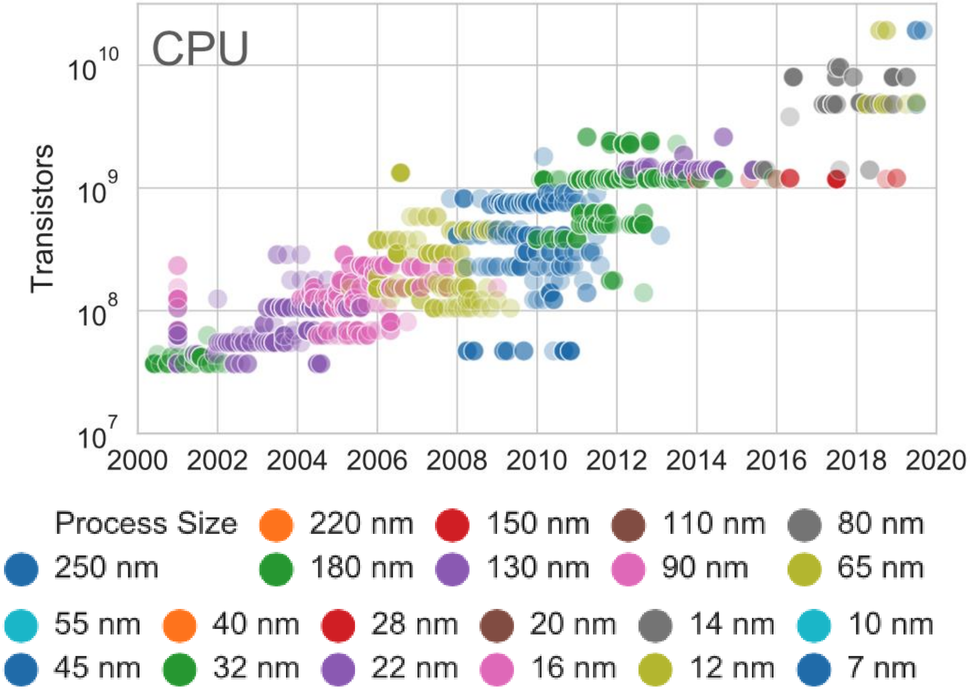
Moore's Law (1965)

- "The number of transistors in a dense integrated circuit will double every two years"

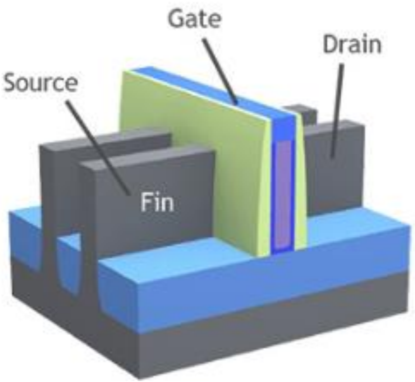


Who is smarter than Gordon Moore?

The End of General Purpose Computing

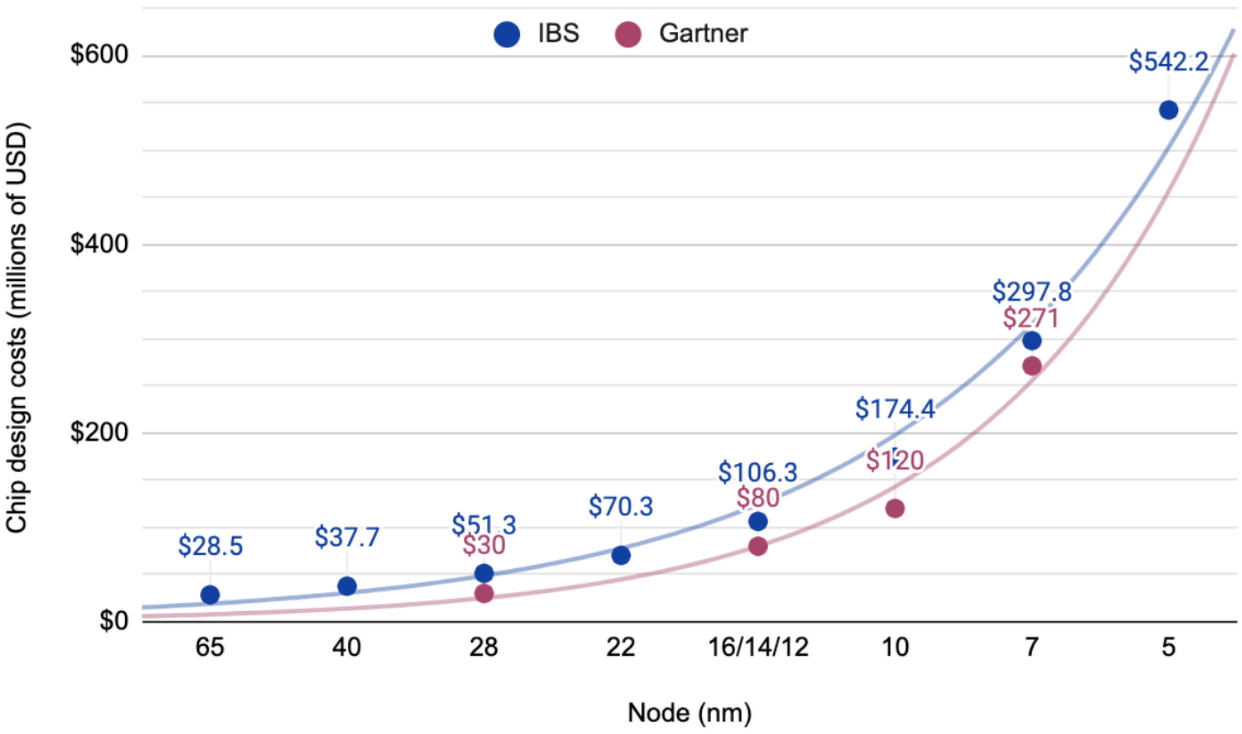


Transistor

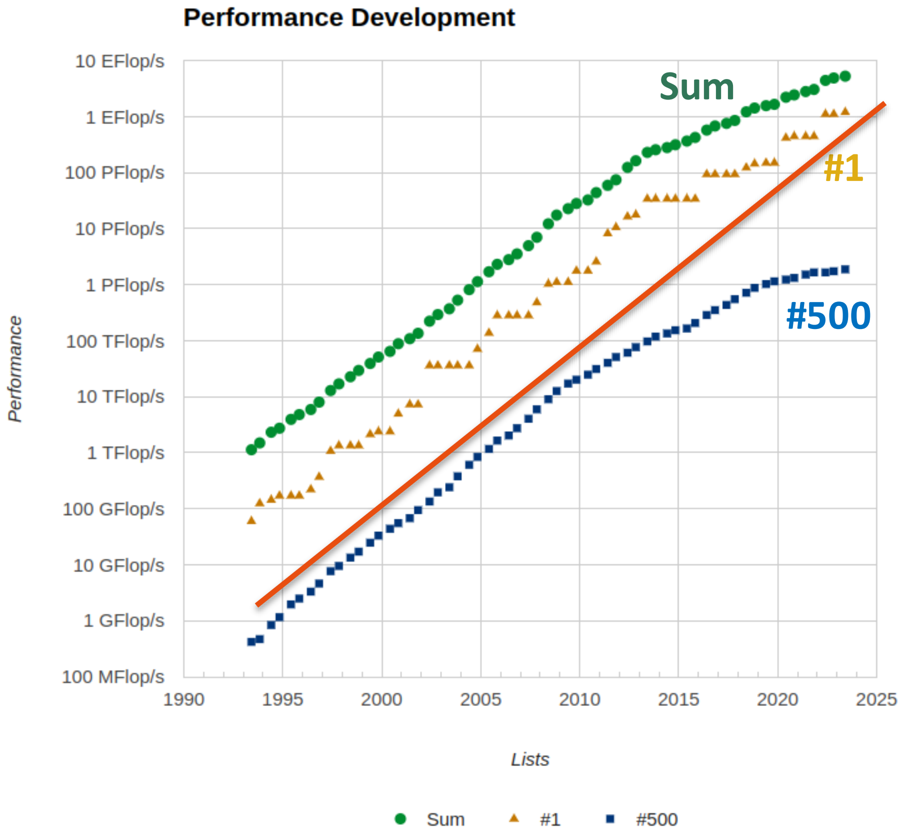


Distance between Si-atoms is 0.5 nm!

Chip Design Costs



How does performance of our machines behave

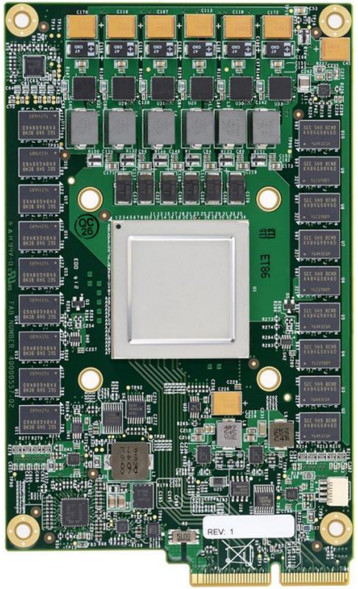


Source: top500.org

So why are we still ok?

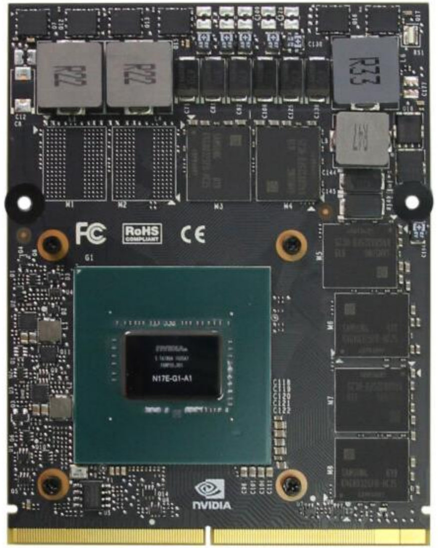
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)	
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703	AMD GPU
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899	
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016	AMD GPU
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,824,768	238.70	304.47	7,404	NVIDIA GPU
5	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096	NVIDIA GPU
6	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94.64	125.71	7,438	NVIDIA GPU
7	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93.01	125.44	15,371	
8	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70.87	93.75	2,589	NVIDIA GPU
9	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63.46	79.22	2,646	NVIDIA GPU
10	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4,981,760	61.44	100.68	18,482	Xeon Phi

Specialized Chips are on the Rise!



Google's TPU
(e.g. machine learning)

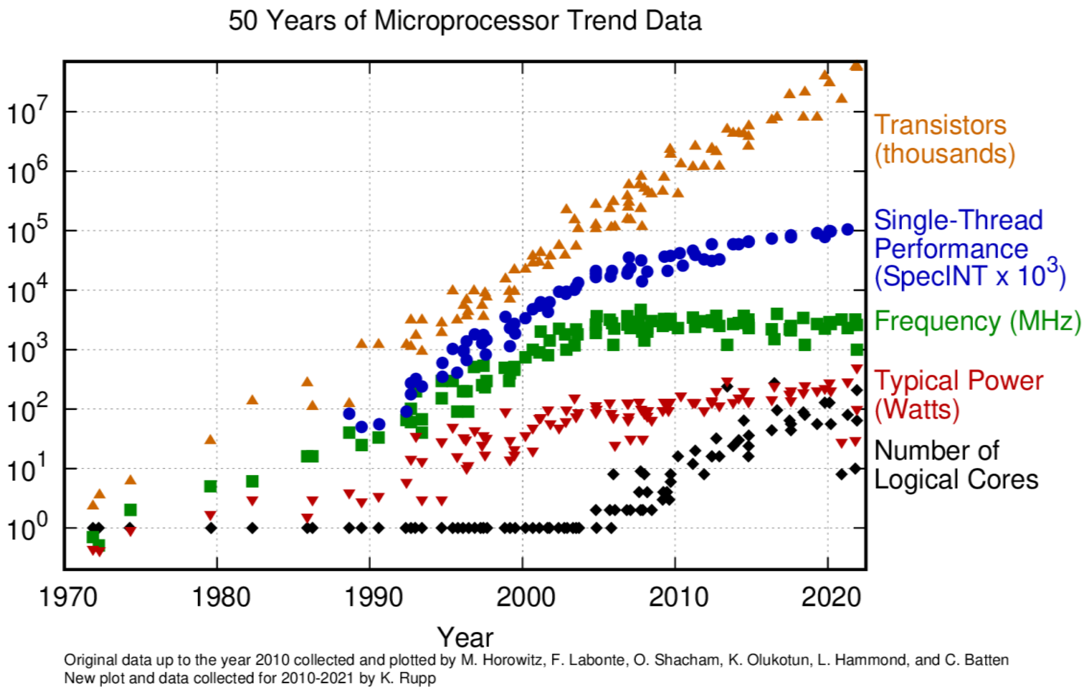
FPGA
(e.g. bitcoin mining)



GPU
(e.g. gaming)

Dennard Scaling

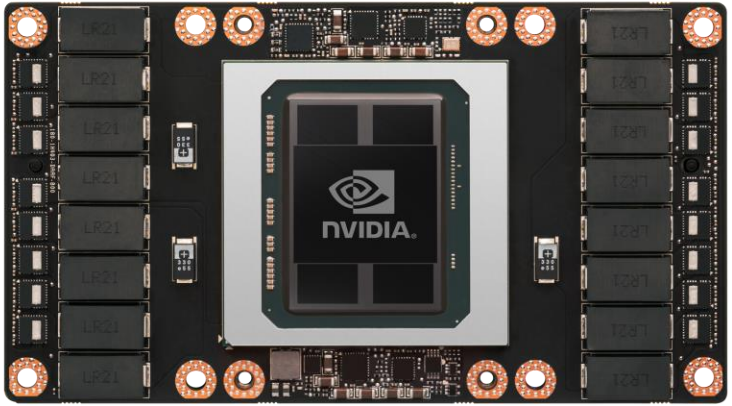
- "If the transistor density doubles, power consumption (with twice the number of transistors) stays the same."



Performance / Watt

Intel Xeon E5-2690 v3 + DRAM

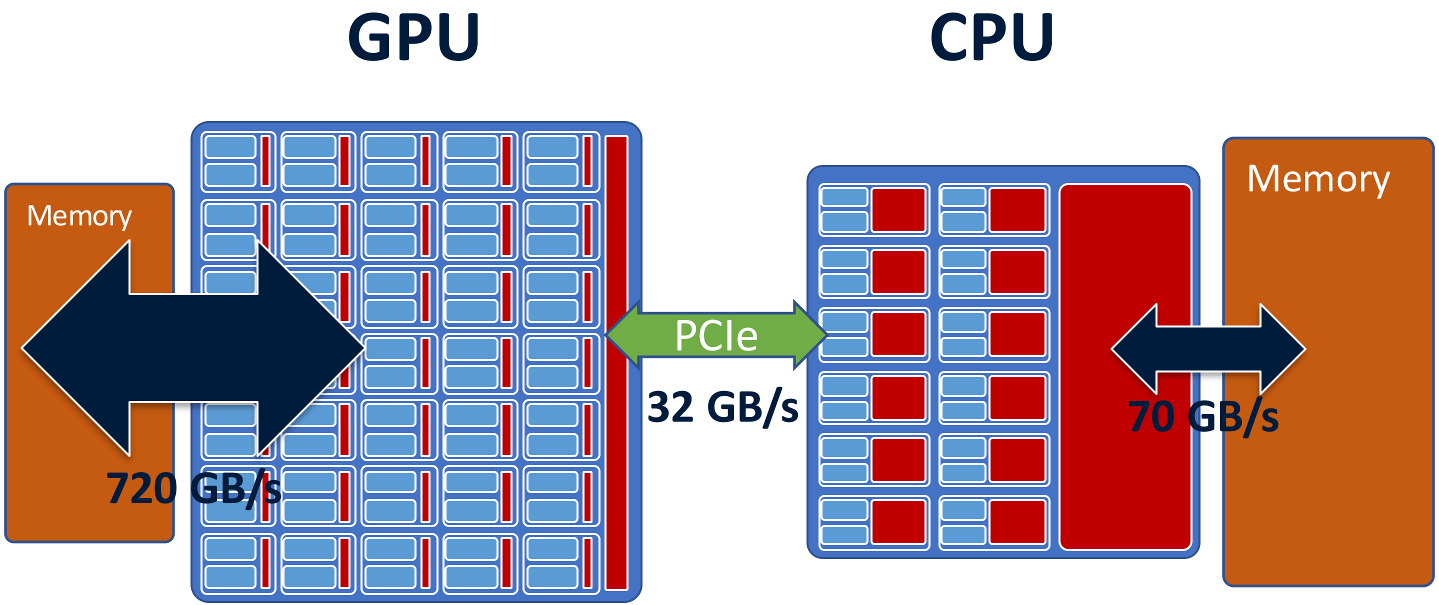
NVIDIA Tesla P100



~ 200 W   0.5 TFLOP/s   70 GB/s

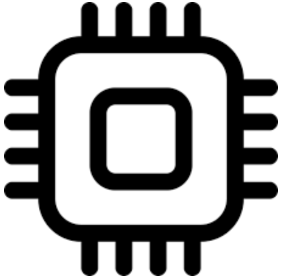
~ 300 W   5.3 TFLOP/s   720 GB/s

Node Architecture

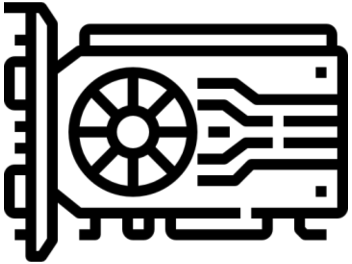


Crucial to minimize memory transfers between CPU and GPU!

CPU vs. GPU



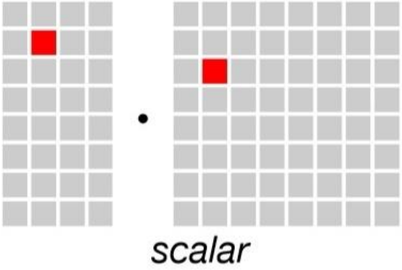
Architecture



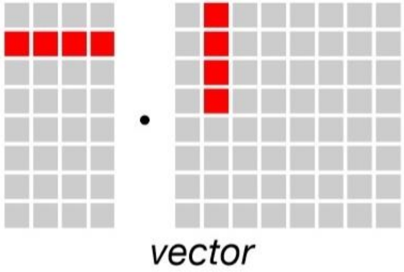
Latency

Optimization

Bandwidth

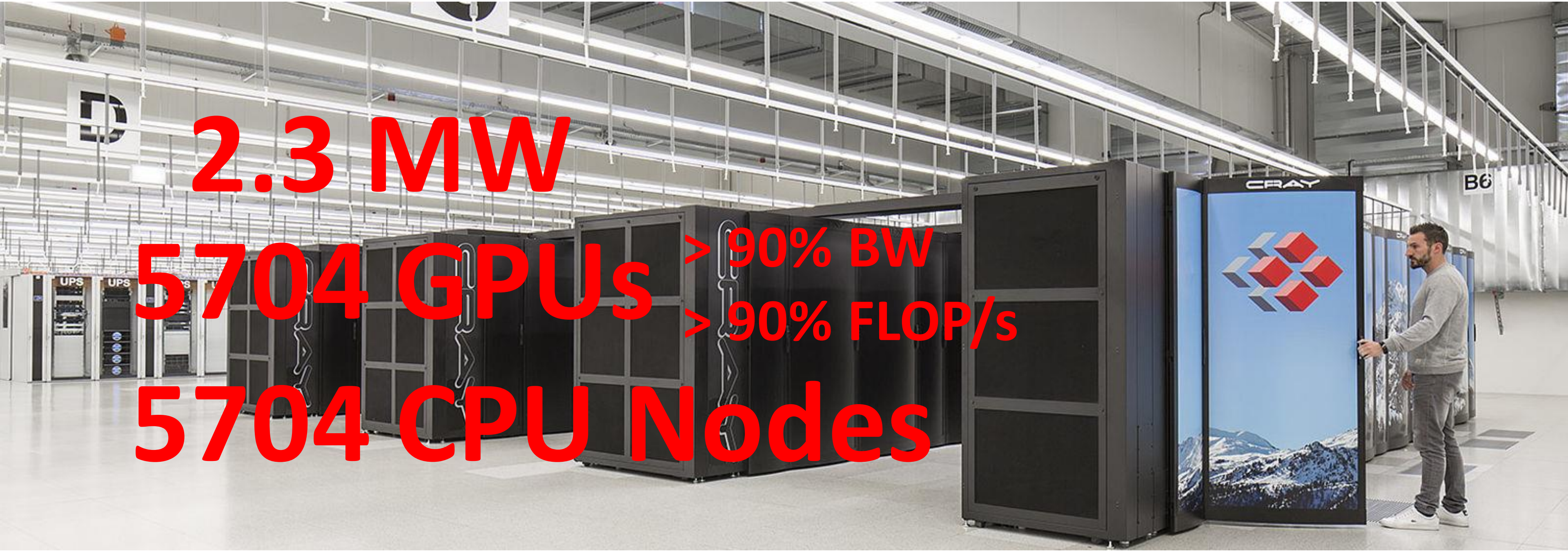


Compute primitive



So how does this connect to weather and climate?

Hybrid Supercomputer



2.3 MW

5704 GPUs

> 90% BW

> 90% FLOP/s

5704 CPU Nodes

Source: www.eia.gov

Power, power, power!

Scalability tests with IFS on Piz Daint for simulations with 1.45km grid spacing (Düben et al., 2020)

Dycore option	#tasks and threads	Energy consumption per year	Throughput
Hydrostatic	4880 tasks; 12 threads per task	85.21 MWh/SY	0.190 SYPD
Non-hydrostatic	9776 tasks; 6 threads per task	191.74 MWh/SY	0.088 SYPD
Non-hydrostatic	4880 tasks; 12 threads per task	195.30 MWh/SY	0.085 SYPD

191.74 MWh / SY * 0.088 SY / day = 16'874 kWh / day

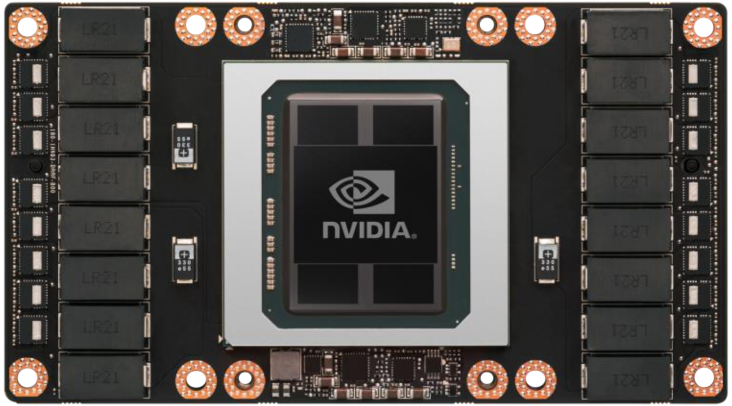
Compare to

Average electricity consumption for one household ~ 29 kWh / day

Performance / Watt

Intel Xeon E5-2690 v3 + DRAM

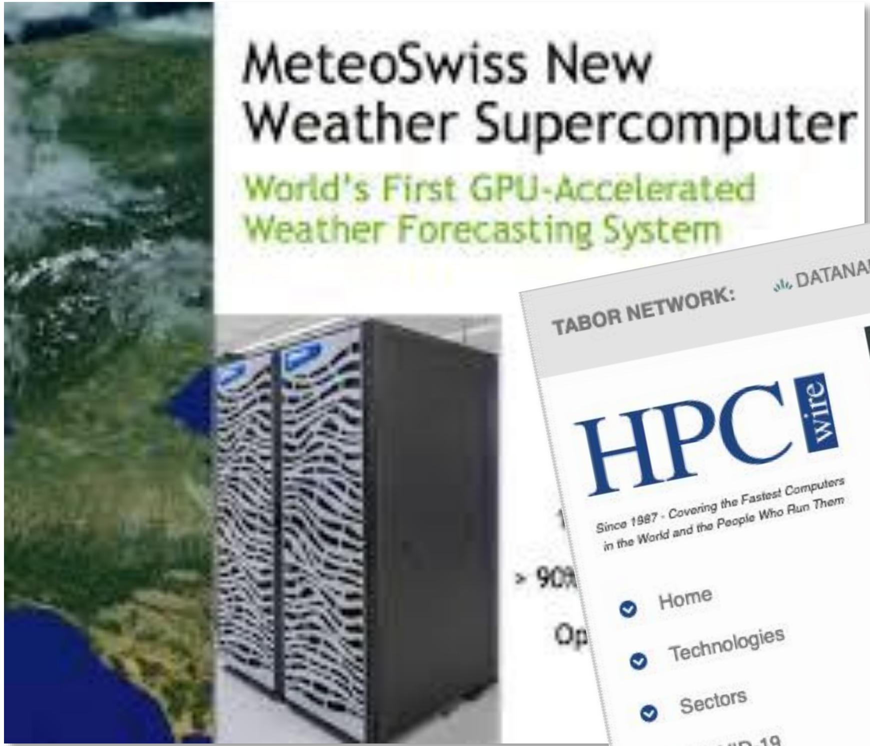
NVIDIA Tesla P100



~ 200 W 0.5 TFLOP/s 70 GB/s

~ 300 W 5.3 TFLOP/s 720 GB/s

Weather and Climate on GPUs



The Demonstration of pace

Simulation Details

Based on a port of NOAA's FV3GFS/X-SHIELD model

Implemented in Python based on a domain-specific language GT4Py

Finite-volume non-hydrostratic dynamics

Global simulation on a cubed-sphere grid with $\Delta x = 1.85$ km

Executed on 4056 GPU-accelerated nodes of the Piz Daint supercomputer

Throughput of 0.071 SYPD (simulated years per day)

The Day 1 Comparison

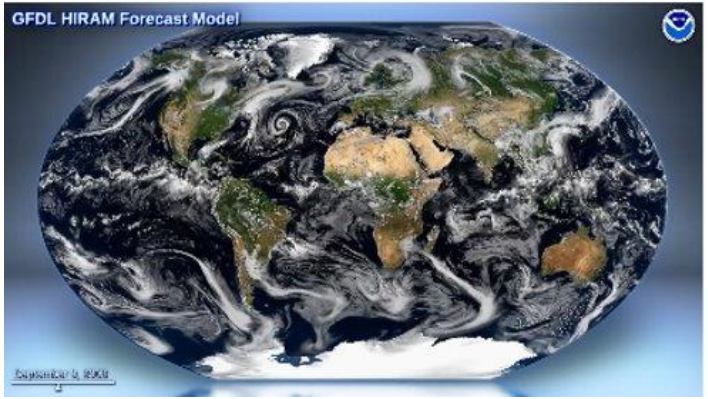
- Global climate model with 3 km resolution (DYAMOND)

  $57 \times 10^6 \times 80$ gridpoints

  $\Delta x = 3.0$ km

  $\Delta t = 9$ s

  1000x faster than real time (3 SYPD)

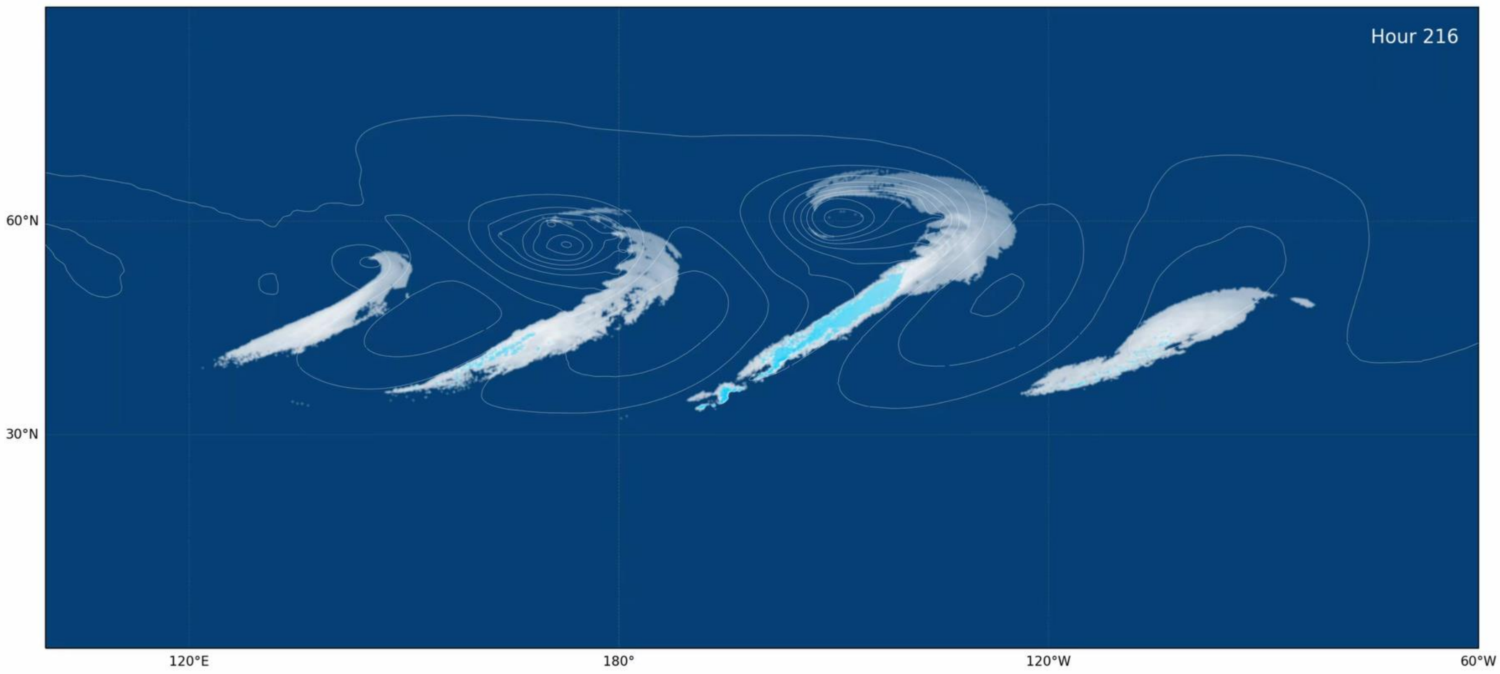


Simulation Details

Based on a port of NOAA's FV3GFS/X-SHIELD model
Implemented in Python based on a domain-specific language GT4Py
Finite-volume non-hydrostatic dynamics
Global simulation on a cubed-sphere grid with $\Delta x = 1.85$ km
Executed on 4056 GPU-accelerated nodes of the Piz Daint supercomputer
Throughput of 0.071 SYPD (simulated years per day)

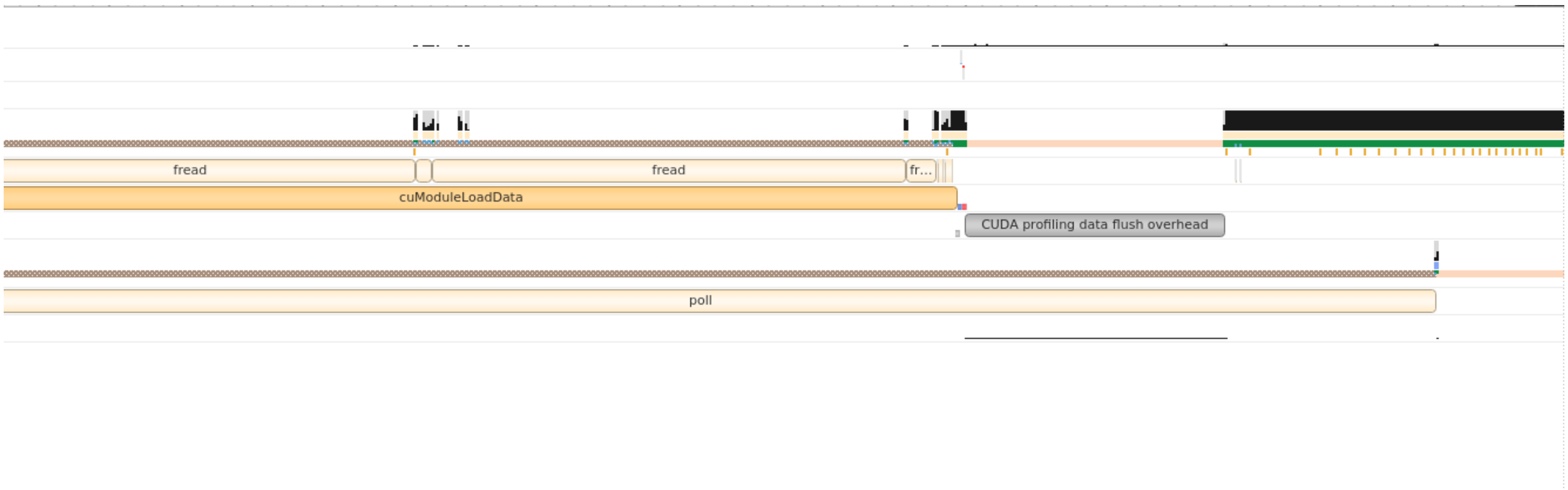
We are still missing 1 order of magnitude

The Demonstration of pace

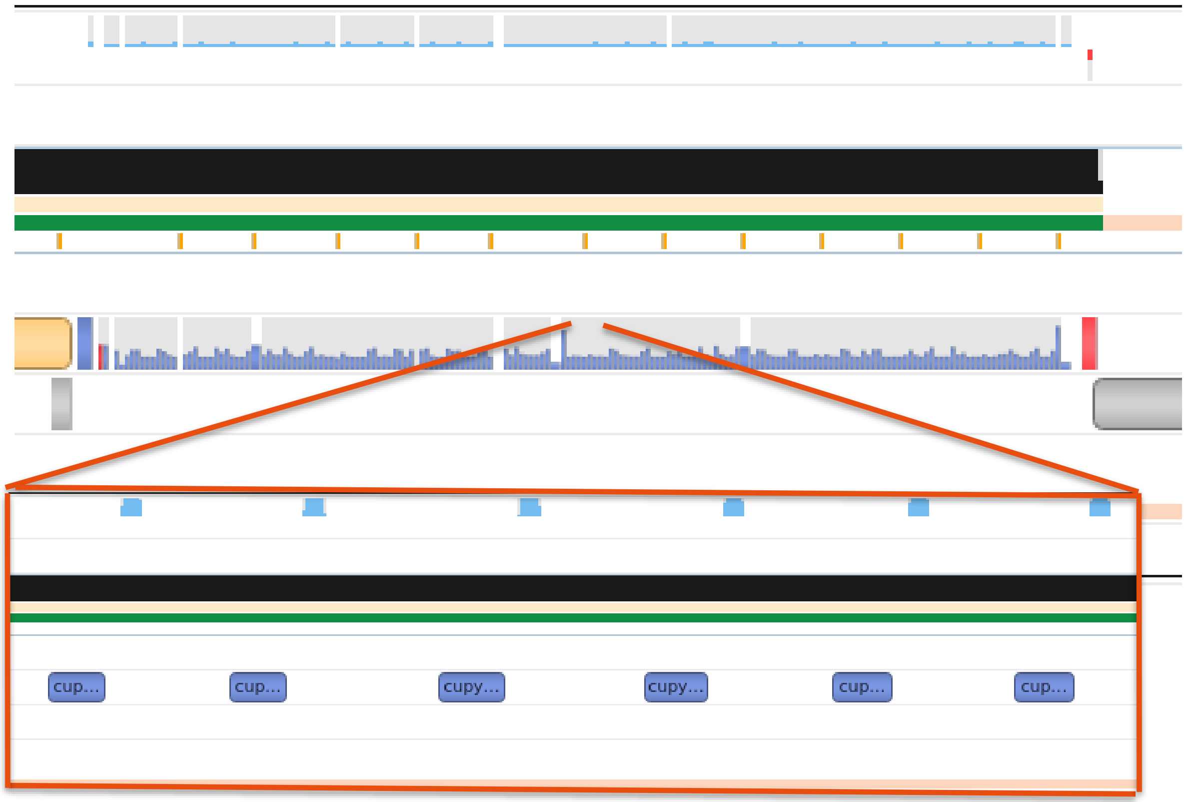


What are the main pitfalls when doing GPU programming?

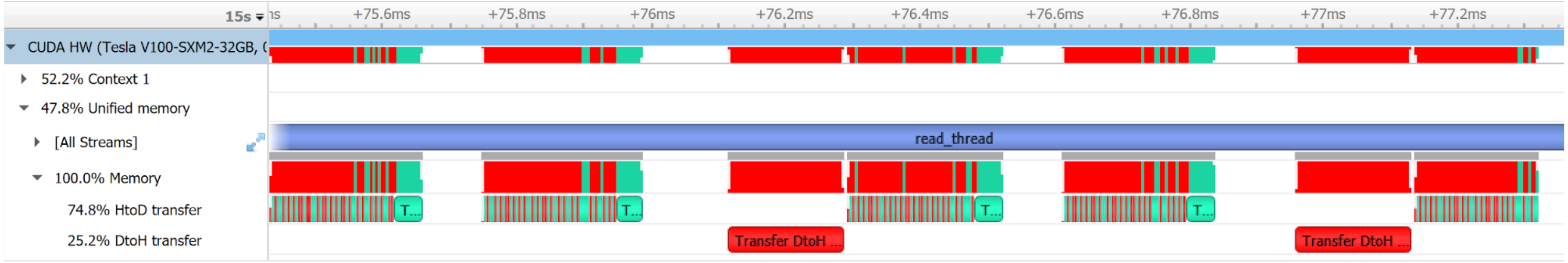
Why GPU Programming is Hard - Work



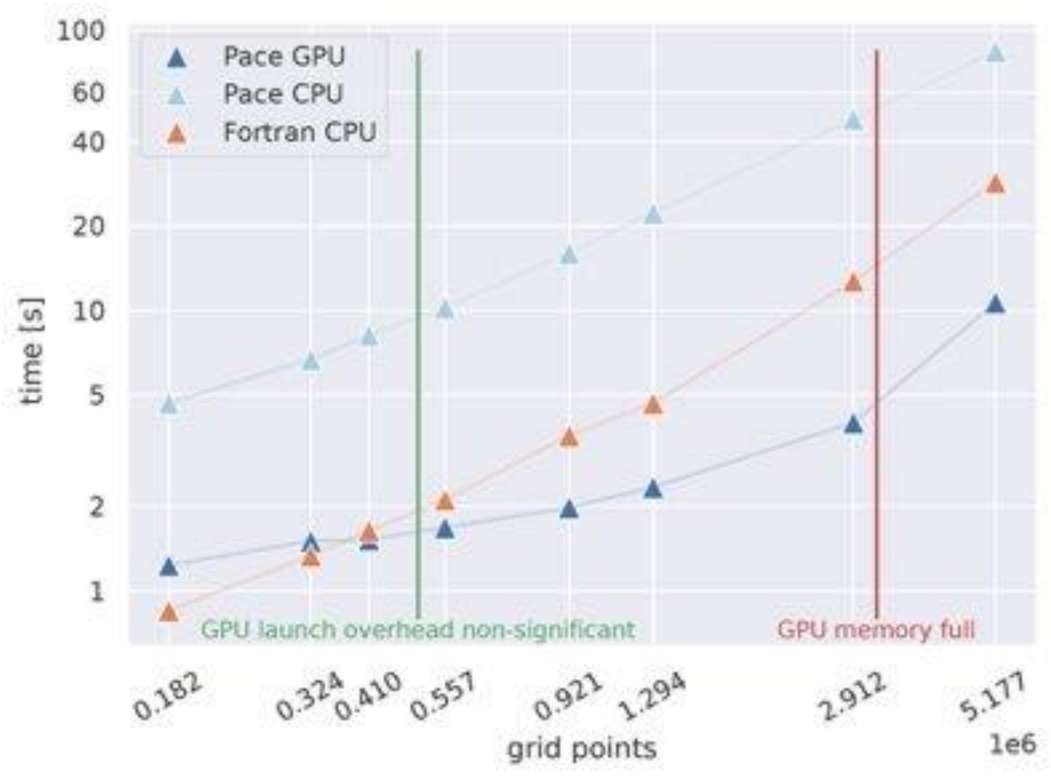
Why GPU Programming is Hard - Calls



The Demonstration of pace - Memory



The Demonstration of pace - Memory



How did you program GPUs?

How to program GPUs

Application

Libraries

CuPy

Directives

OpenACC

Directives for Accelerators

Programming Languages

NVIDIA

CUDA

How is it done?

```
!$acc enter data async create (this)

!fixed size arrays allocation
!$acc enter data async create (this%c_reff_so, this%c_
!$acc enter data async create (this%c_rhoquer, this%c_


!allocate pointers
!$acc enter data async create (this%r_eff_so, this%r_e
!$acc enter data async create (this%r_

!$acc data present(this)

!$acc parallel async
!$acc loop gang vector collapse (3)
DO k= 1 , ke
  DO i= 1, nproma
    DO j= 1, nhabits_ice
      this%r_eff_so(i,k,j)     = 0.0_w
      this%r_eff_th(i,k,j)     = 0.0_w
      this%ar_mean_th(i,k,j)   = 0.0_w
    END DO
  END DO
END DO
!$acc end parallel
```

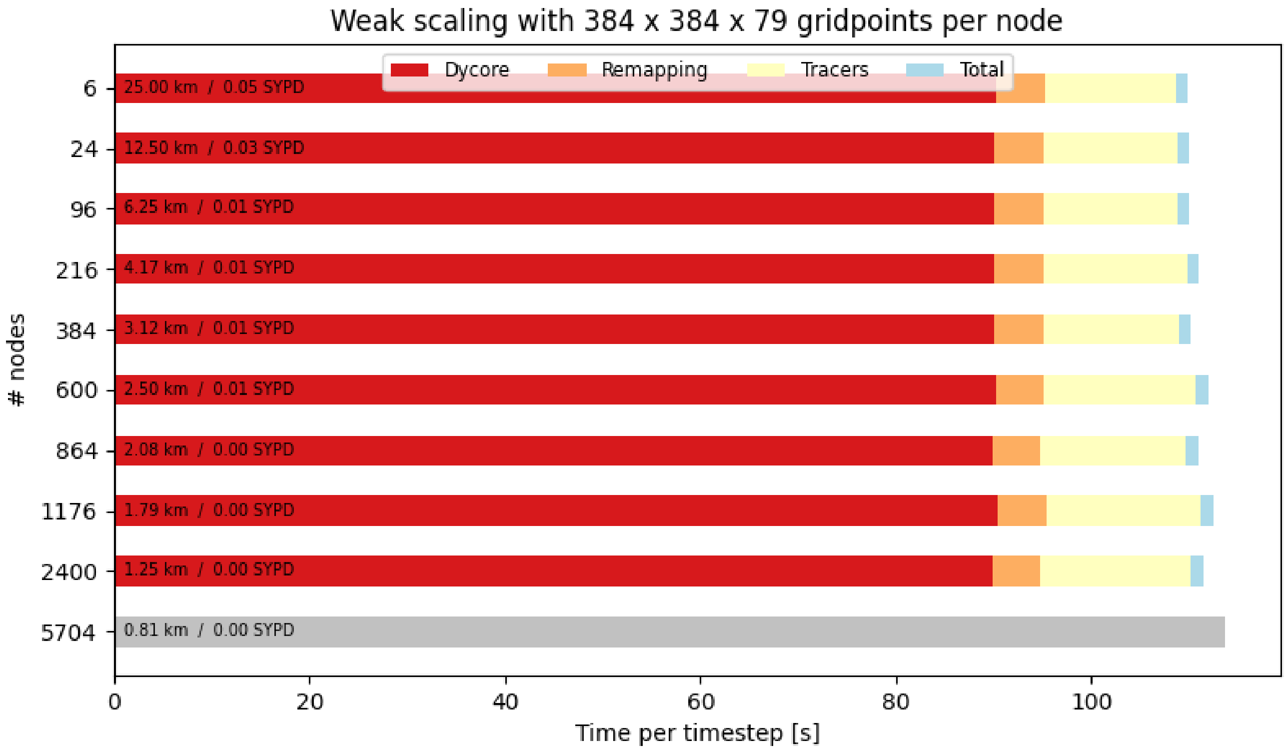
```
# reset fields
with computation(FORWARD), interval(...):
    zfix = 0
    sum0 = 0.0
    sum1 = 0.0
with computation(PARALLEL), interval(...):
    lower_fix = 0.0
    upper_fix = 0.0
# fix_top:
with computation(BACKWARD):
    with interval(1, 2):
        if q[0, 0, -1] < 0.0:
            q = (
                q + q[0, 0, -1] * dp[0, 0, -1] / d
            )  # move enough mass up so that the t
    with interval(0, 1):
        if q < 0:
            q = 0
        dm = q * dp
# fix_interior:
with computation(FORWARD), interval(1, -1):
    # if a higher layer borrowed from this one, account for that here
    if lower_fix[0, 0, -1] != 0.0:
        q = q - (lower_fix[0, 0, -1] / dp)
    if q < 0.0:
        zfix += 1
        if q[0, 0, -1] > 0.0:
            # Borrow from the layer above
```

```
pack_scalar_f64_kernel = (
    None
    if cp is None
    else cp.RawKernel(
        r"""
        extern "C" __global__
        void pack_scalar_f64(const double* i_sourceArray,
                             const int* i_indexes,
                             const int i_nIndex,
                             const int i_offset,
                             double* o_destinationBuffer)
        {
            int tid = blockDim.x * blockIdx.x + threadIdx.x;
            if (tid>=i_nIndex)
            {
                return;
            }

            o_destinationBuffer[i_offset+tid] = i_sourceArray[i_indexes[tid]];
        }

        """,
        "pack_scalar_f64",
    )
)
```

Why is it done?



How to program GPUs

Libraries

CuPy

Directives

OpenACC

Directives for Accelerators

Programming Languages

NVIDIA

CUDA

Lab Exercises

01-GPU-programming-cupy.ipynb

- Introduction to GPU programming using a high-level programming language

Remarks

- When running a GPU notebook, you may experience this error:

  cupy_backends.cuda.api.runtime.CUDARuntimeError: cudaErrorDevicesUnavailable: all CUDA-capable devices are busy or unavailable

  Don’t worry, it’s not your fault! Just restart the kernel and the error should disappear. If it persists, reach out to us.

- To let multiple tasks access the same GPU: export CRAY_CUDA_MPS=1

Let’s go!