CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Scalability of the Julia/GPU stack

JuliaCon 2022, Minisymposium Julia for HPC

Samuel Omlin[1,2], Ludovic Räss[2,3], Ivan Utkin[2,3]

July 26th 2022

[1] CSCS - Swiss National Supercomputing Centre | [2] ETH Zurich | [3] Swiss Federal Institute for Forest, Snow and Landscape Research (WSL)

Performance

Portability

Productivity

CSCS ETH zürich

(scalable) Performance

(performance) Portability

Productivity

(scalable) Performance

(performance) Portability

Productivity

(scalable) Performance

(performance) Portability

Productivity

Algorithms?

Performance evaluation?

Programming language(s)?

Single node performance?

Distributed parallelization?

(scalable) Performance

(performance) Portability

Productivity

Our solution

Distributed parallelization?

(scalable) Performance

(performance) Portability

Productivity

Our solution

Distributed parallelization?

automatic distributed parallelization
of architecture-agnostic code

The 3 P are tightly connected!

CSCS

ETH zürich

(scalable) Performance

(performance) Portability

Productivity

Our solution

Distributed parallelization?

ImplicitGlobalGrid.jl

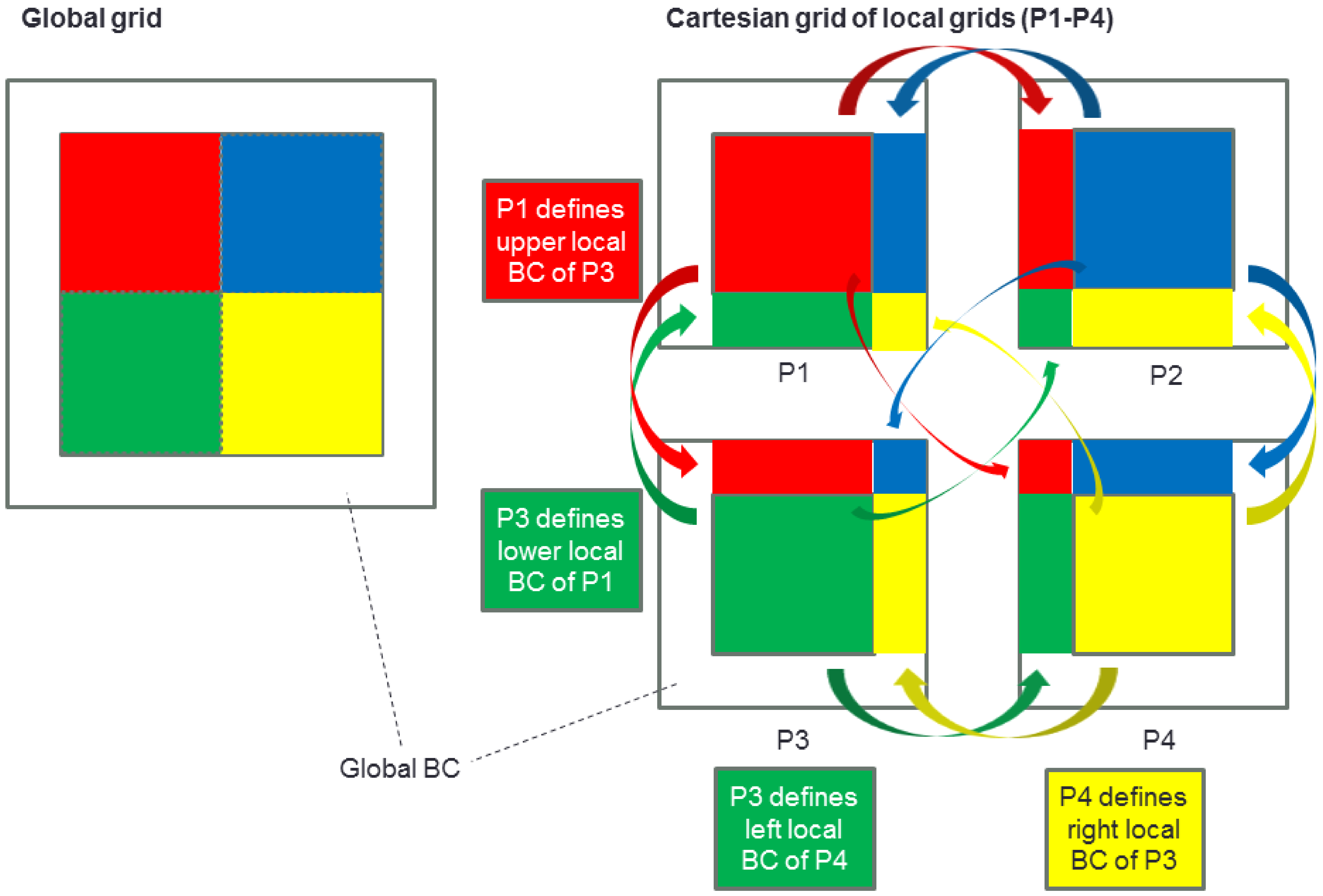
Agenda

- Introduction: the three P ✓
- Automatic distributed parallelization of architecture-agnostic code
- Results & Conclusions

Example: ImplicitGlobalGrid

- for domain scientists
- renders the distributed parallelization of stencil-based GPU and CPU applications on a regular (staggered) grid nearly trivial
- relies on `MPI.jl`, `CUDA.jl` and `AMDGPU.jl` (supports CUDA- and ROCm-aware MPI)
- seamlessly interoperable with `MPI.jl` and of course also with `ParallelStencil.jl`.
- uses implicit domain decomposition

Multi-GPU – implicit domain decomposition?



Automatic distributed parallelization of architecture-agnostic code – using ImplicitGlobalGrid (+ ParallelStencil) (1/2)

```
const USE_GPU = true
using ParallelStencil
using ParallelStencil.FiniteDifferences2D
@static if USE_GPU
    @init_parallel_stencil(CUDA, Float64, 2)
else
    @init_parallel_stencil(Threads, Float64, 2)
End
using ImplicitGlobalGrid

#(…)

me, dims, nprocs = init_global_grid(nx, ny, 1)
dx = lx/(nx_g()-1)
dy = ly/(ny_g()-1)
```

Automatic distributed parallelization of architecture-agnostic code – using ImplicitGlobalGrid (+ ParallelStencil) (2/2)

```
# Time loop
t=0.0;
for it = 1:nt
    H_t .= H
    err=2.0*tolnl; iter=1
    while err > tolnl
        if (iter % ncheck == 0) H_τ .= H; end
        @parallel compute_flux!(qHx, qHy, H, B, g_mu, dx, dy)
        @parallel update_H!(H, dHdτ, qHx, qHy, g_mu, dx, dy, dt)
        update_halo!(H);
        if (iter % ncheck == 0) err = mean(abs.((H.-H_τ))); end
        iter+=1;
    end
    t = t + dt
end

finalize_global_grid()
```

Automatic distributed parallelization of architecture-agnostic code – using ImplicitGlobalGrid (+ ParallelStencil) (2/2)

```
# Time loop
t=0.0;
for it = 1:nt
    H_t .= H
    err=2.0*tolnl; iter=1
    while err > tolnl
        if (iter % ncheck
        @parallel compute_
        @parallel update_H
        update_halo!(H);
        if (iter % ncheck
        iter+=1;
    end
    t = t + dt
end

finalize_global_grid()
```

Array **H** can be an Array, CuArray or ROCArray – multiple dispatch!

No need for explicit send / receive buffer or CUDA stream / ROCm queue allocation:

- Automatic allocation at first **`update_halo!`** call, then reuse until **`finalize_global_grid`**
- Reallocation of more memory only if bigger/more arrays

Requirements of `update_halo!` on MPI.jl and the Julia GPU stack

Communication via host

- **page-locking** of host memory (for maximal device<->host transfer speed)
- **mapping** of host memory to device (for access from GPU kernel)
- CUDA-/ROCm-provided **optimized async routines** for e.g. 3-D device<->host memcopy

Communication with RDMA

- RDMA enabled **CUDA-/ROCm-aware MPI**
- (abstract) **device arrays as arguments** for MPI.jl routines

Automatic distributed parallelization of architecture-agnostic code – using ImplicitGlobalGrid (+ ParallelStencil) (2/2)

```
# Time loop
t=0.0;
for it = 1:nt
    H_t .= H
    err=2.0*tolnl; iter=1
    while err > tolnl
        if (iter % ncheck ==
        @parallel compute_flux!(qHx, qHy, H, B, g_mu, dx, dy)
        @hide_communication (32, 2) begin
            @parallel update_H!(H, dHdτ, qHx, qHy, g_mu, dx, dy, dt)
            update_halo!(H);
        end
        if (iter % ncheck == 0) err = mean(abs.((H.-H_τ))); end
        iter+=1;
    end
    t = t + dt
end

finalize_global_grid()
```

Automatically hide communication behind computation (only for GPU to date):

1. Compute boundary region (here of size 32 in x dimension, 2 in y dimension) and inner points on two different streams.
2. As soon as boundary region has finished, start halo update

Requirements of @`hide_communication` on MPI.jl and the Julia GPU stack

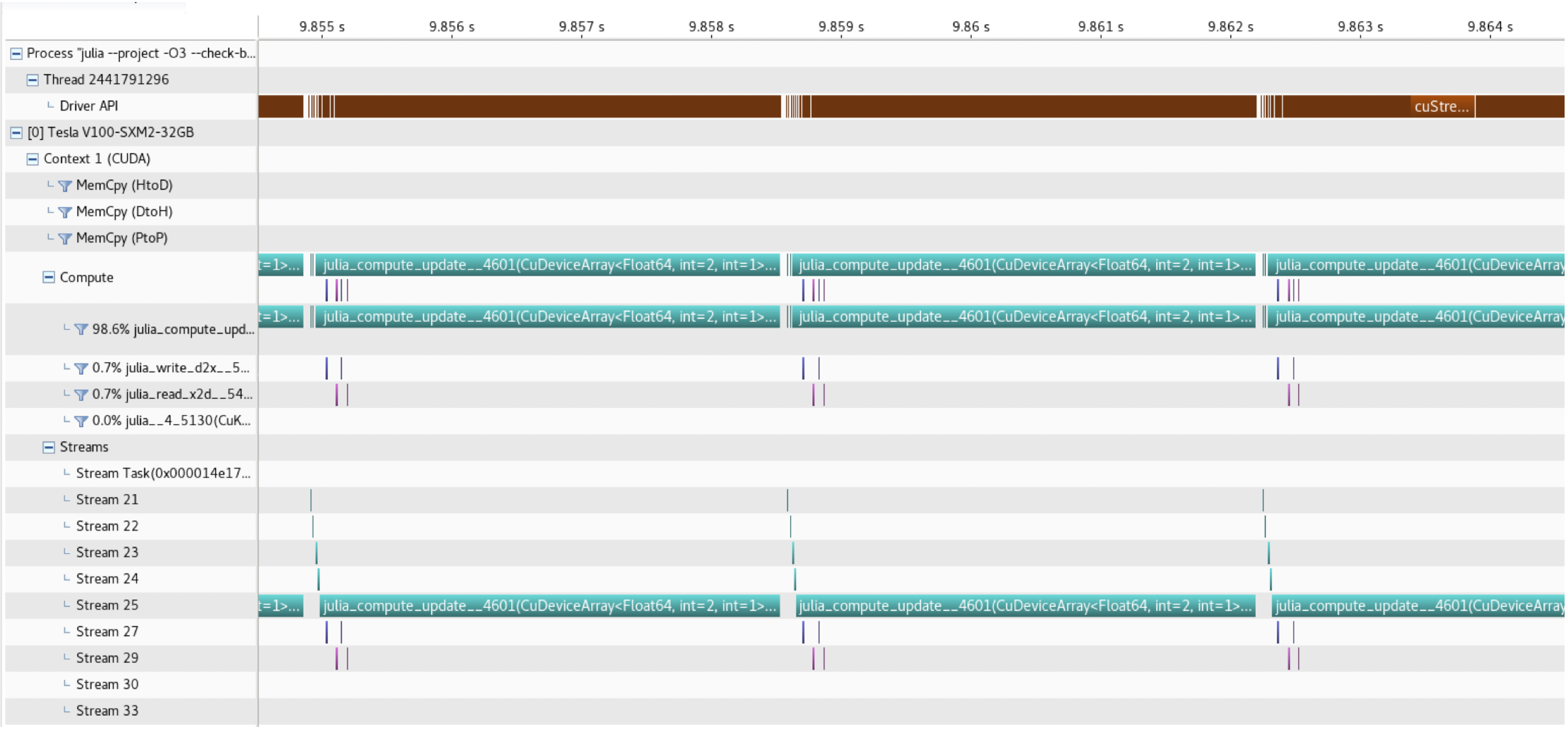
Communication via host

- **Nonblocking** high-priority streams (for overlapping access of **host** send/recv buffers with computations)
- (**Nonblocking**) CUDA-/ROCm-provided optimized **async** routines for e.g. 3-D device<->host memcopy

Communication with RDMA

- **Nonblocking** high-priority streams (for overlapping access of **device** send/recv buffers with computations)
- (**Nonblocking** RDMA data transfer routines in MPI library)

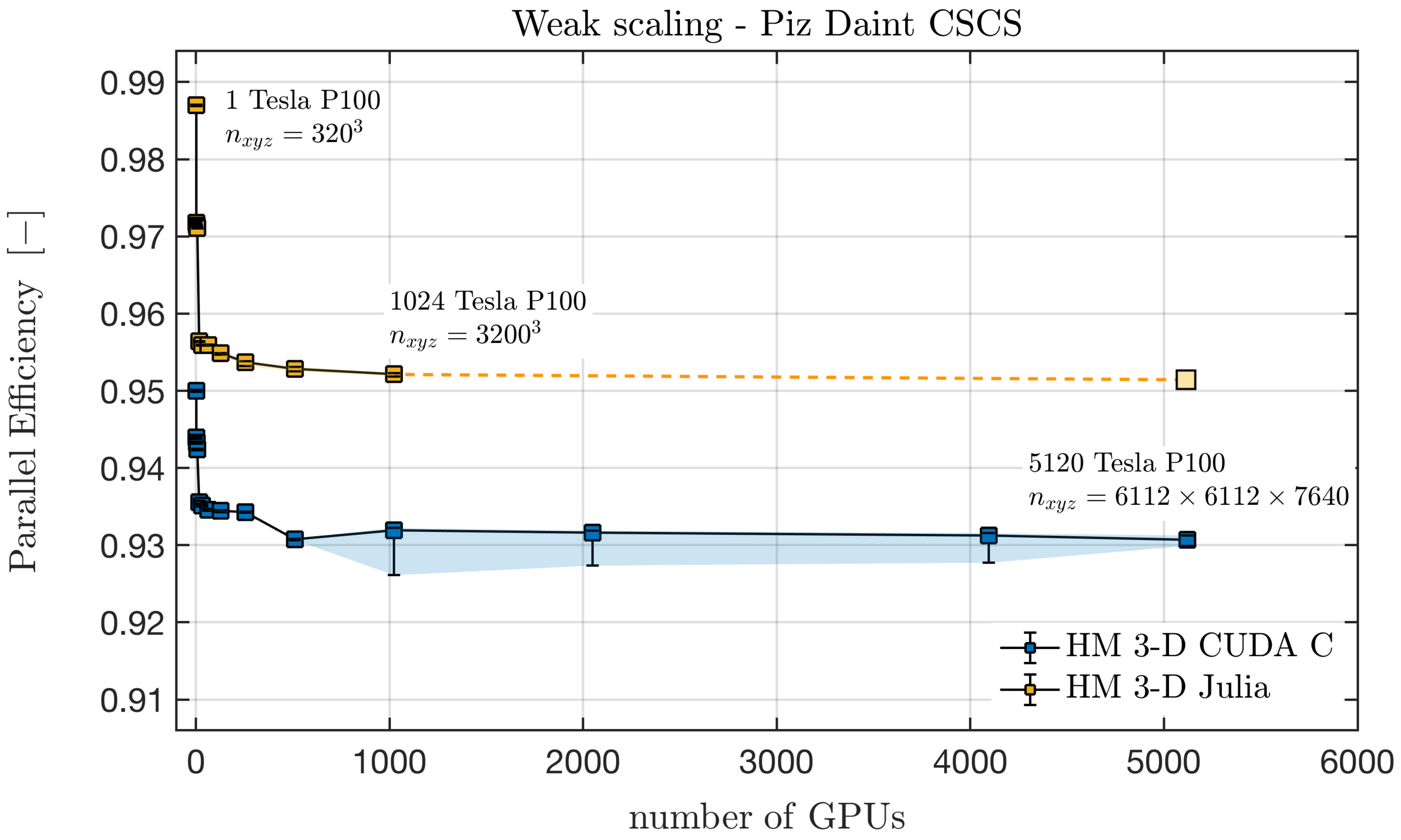
Hiding of communication behind computation



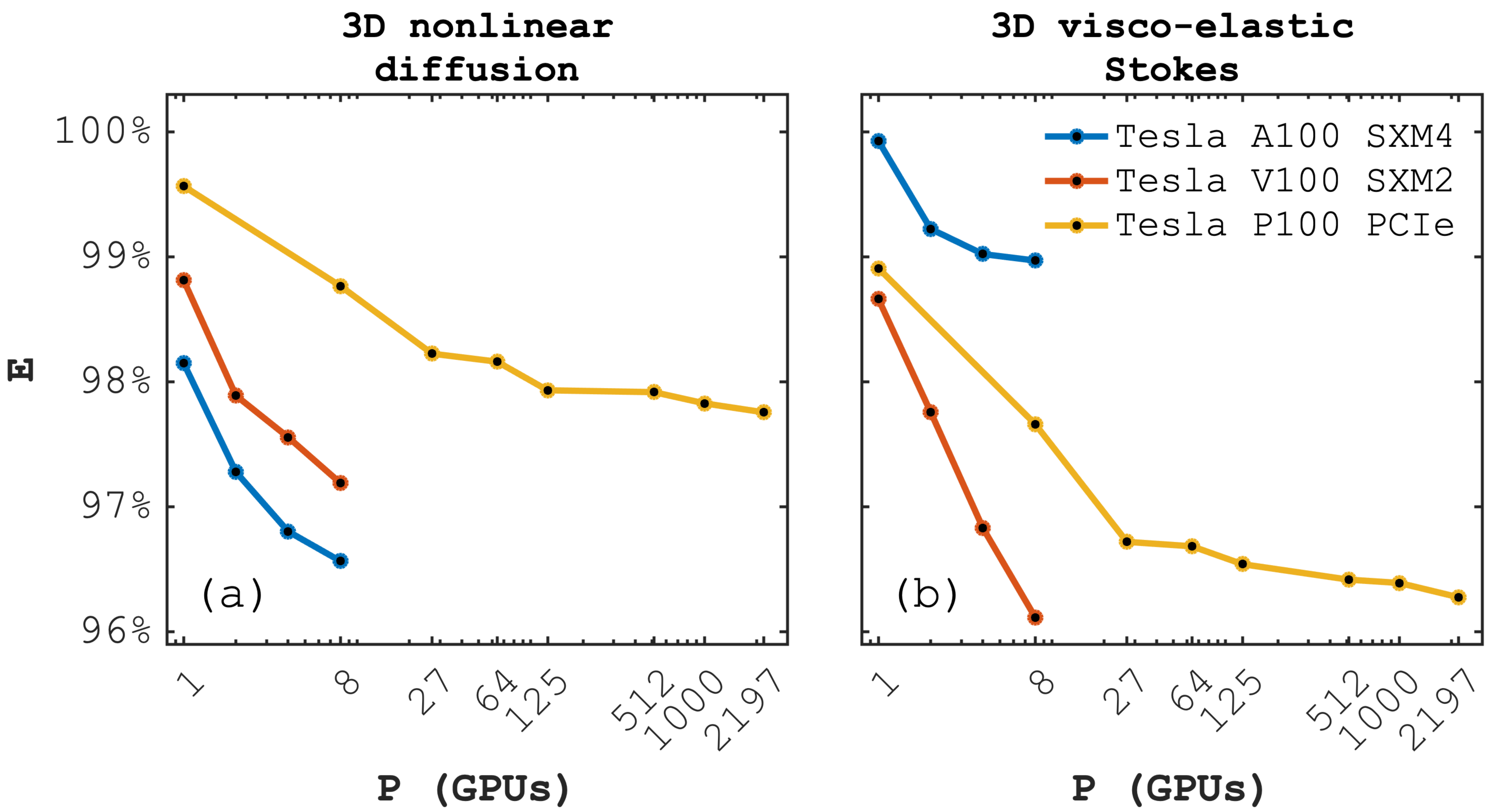
Agenda

- Introduction: the three P ✓
- Automatic distributed parallelization of architecture-agnostic code ✓
- Results & Conclusions

Scaling results: 3-D Poro-visco-elastic twophase flow



Scaling results: 3-D geodynamic building blocks



Conclusions

(scalable) Performance

(performance) Portability

Productivity

Our solution

Algorithms?

Performance evaluation?

julia

Single node performance?

ImplicitGlobalGrid.jl

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Thank you for your kind attention

Side note

Information, Material and edited video of Julia GPU HPC course at CSCS (instructors: Tim Besard and myself):

https://github.com/omlins/julia-gpu-course