Porting a parallel rotor wake simulation to GPGPU accelerators using OpenACC

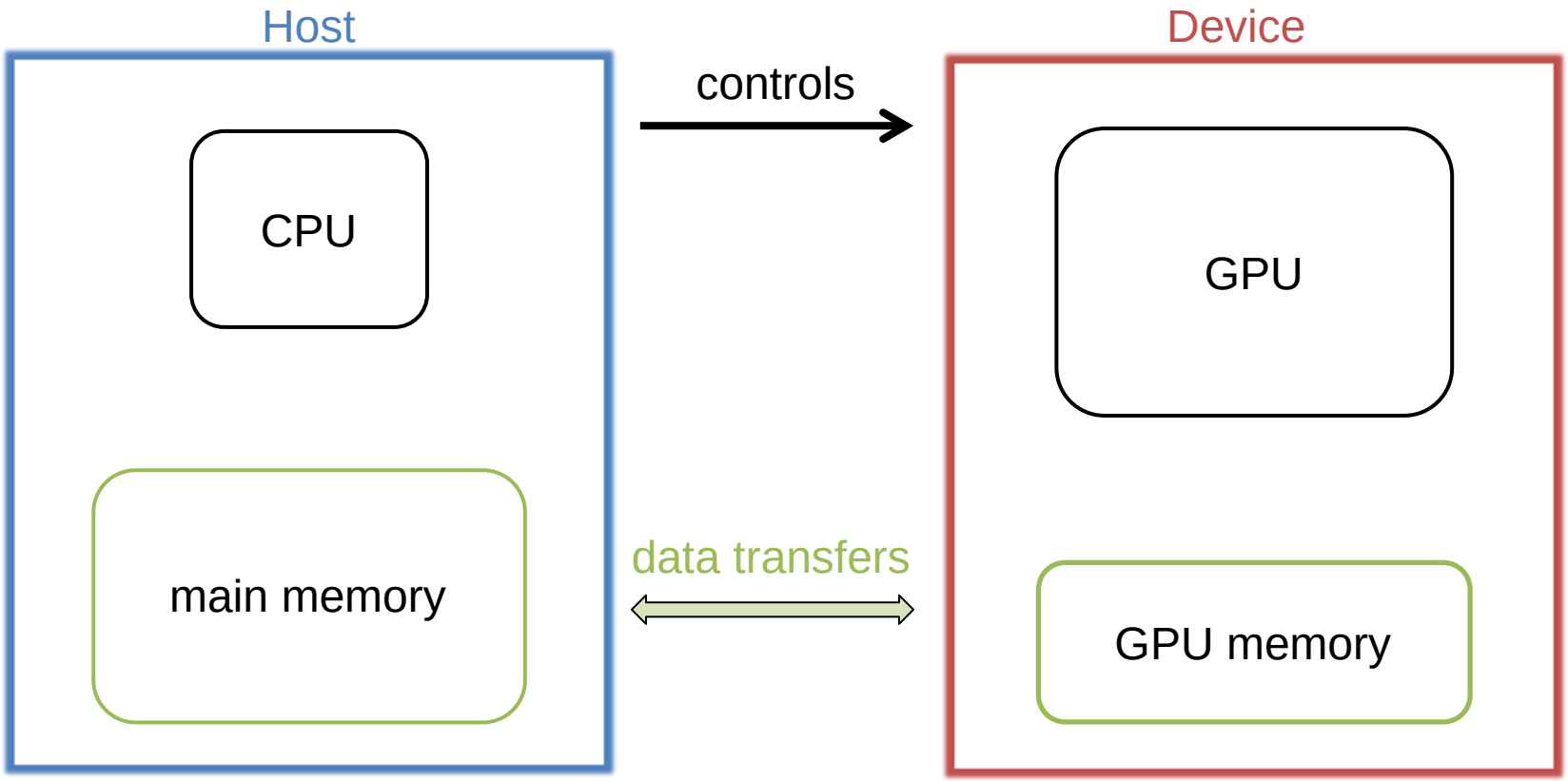
Melven Röhrig-Zöllner

DLR, Simulations- und Softwaretechnik

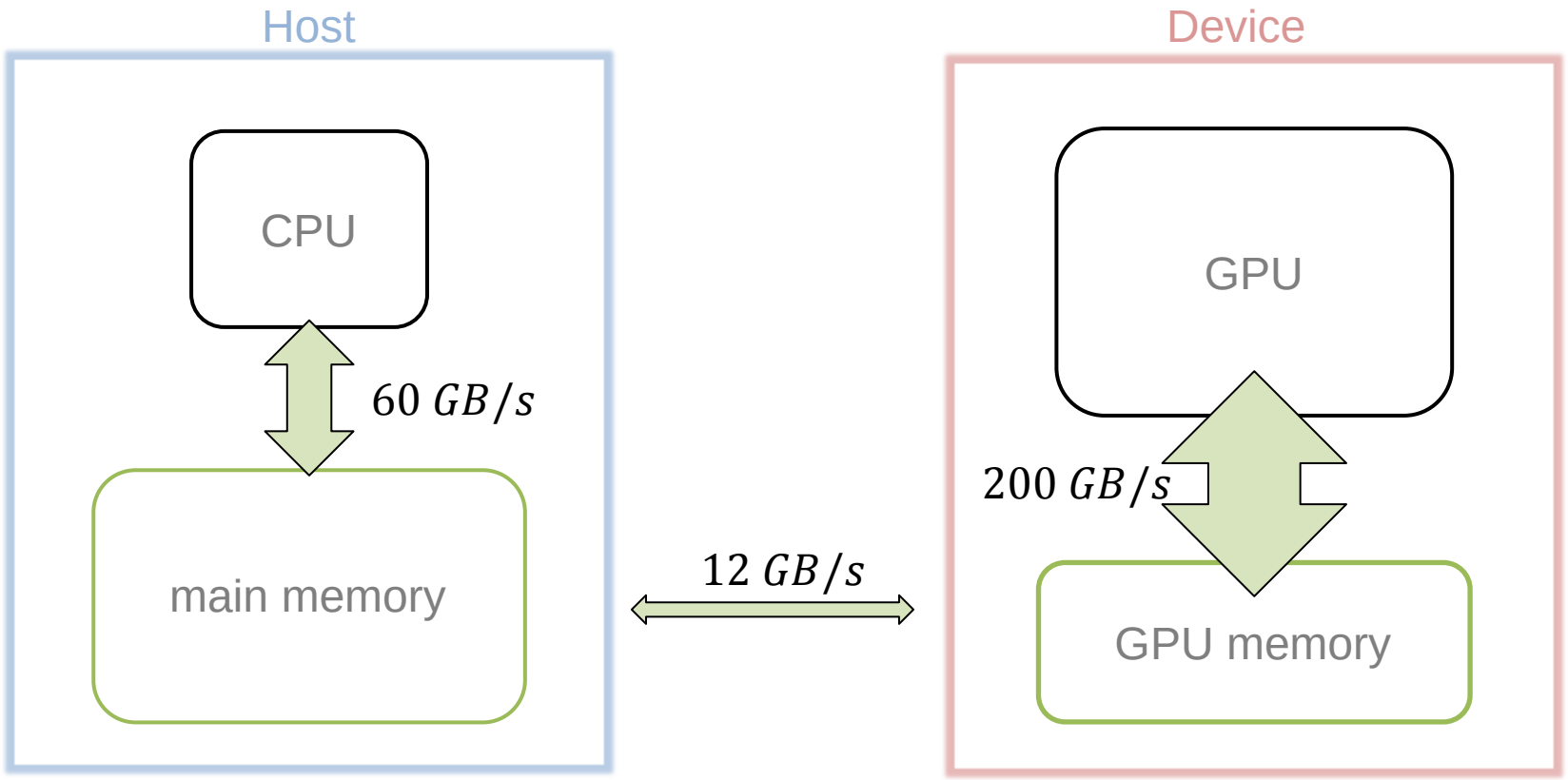
Outline

- **Hardware-Architecture (CPU+GPU)**
- GPU computing with OpenACC
- Rotor simulation code Freewake
- OpenACC port of Freewake
- Conclusion

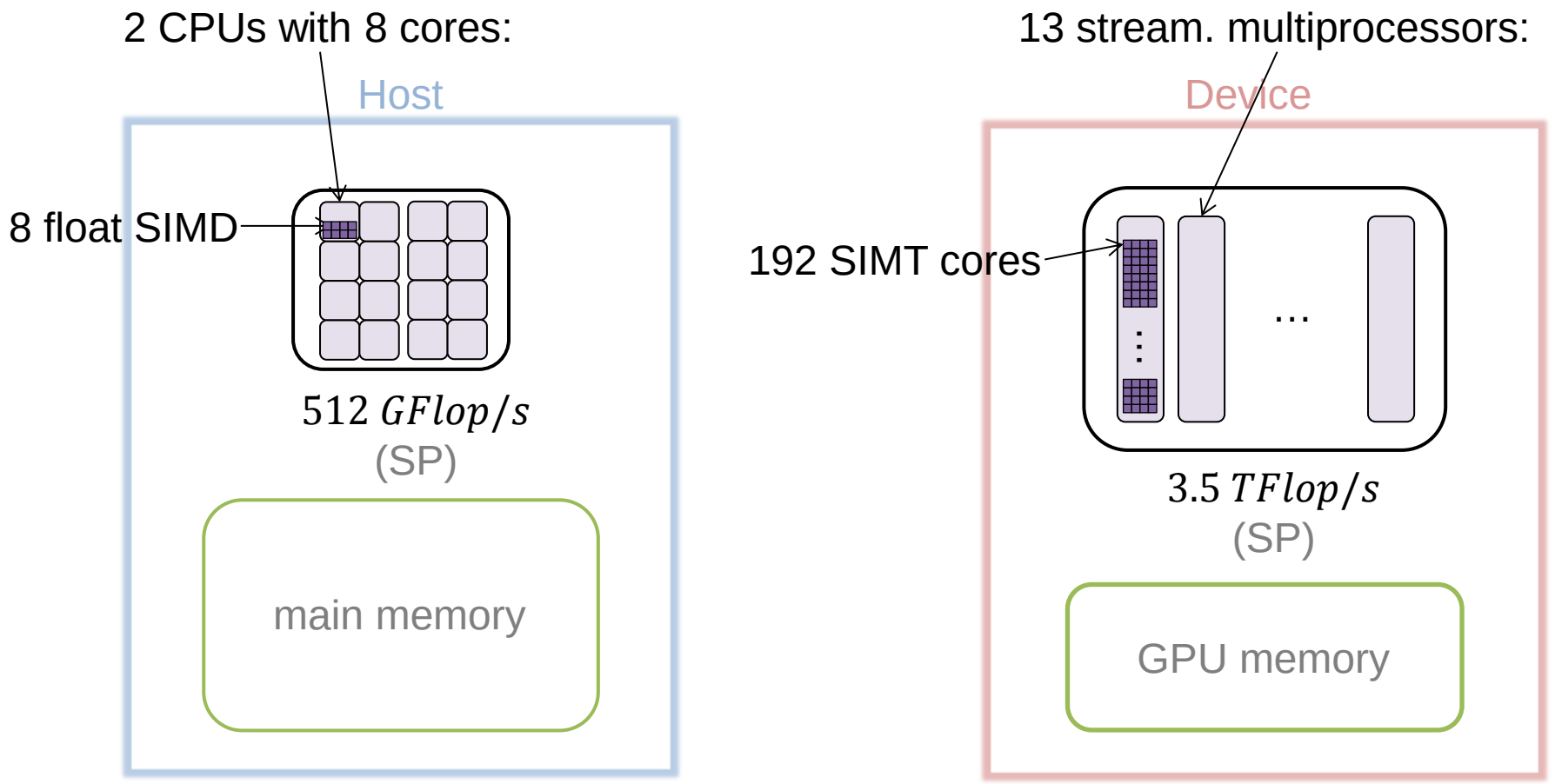
Hardware-Architecture: Overview



Hardware-Architecture: Data transfers



Hardware-Architecture: Calculations



Hardware-Architecture: Comparison

CPU

- SIMD parallelism:
  - SSE / AVX extensions (8 float)
- MIMD parallelism:
  - several CPUs (2)
  - multiple cores (8)
  - (possibly CPU threads)
- **Caches to avoid memory latency**

GPU

- SIMT parallelism:
  - 32 scalar threads form a warp
  - 1-32 warps form a thread block
    (32-1024 threads per block)
- MIMD parallelism:
  - thread blocks within a grid
- **Switch threads to hide latency**

→ Requires 100,000+ threads!

Outline

- Hardware-Architecture (CPU+GPU)
- **GPU computing with OpenACC**
- Rotor simulation code Freewake
- OpenACC port of Freewake
- Conclusion

OpenACC: Overview

- Language extensions similar to OpenMP
- Directive based
- Supported languages:
  - C
  - C++
  - **Fortran**
- Explicit data movement between host and GPU (bottleneck!)
- Supported compilers:
  - CAPS
  - CRAY
  - **PGI**
  - (unofficial patches for GCC)

OpenACC: Example

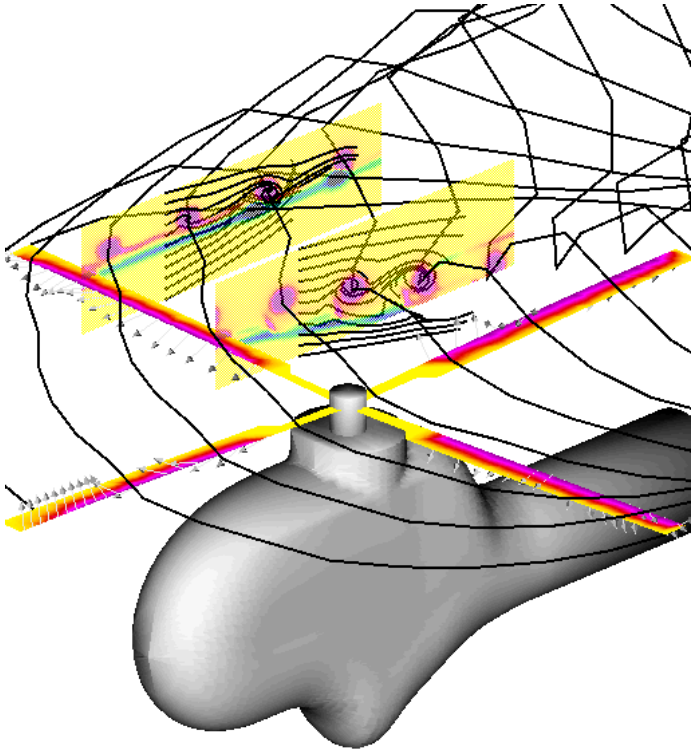
```
program main
    real :: a(N)
    …
!$acc data copyout(a)
    ! Computation in several loops on the GPU:
    …
    !$acc parallel loop
    do i = 1, N
        a(i) = 2.5 * i
    end do

    …
!$acc end data
    ! Use results on the CPU
    …
end program main
```

Outline

- Hardware-Architecture (CPU+GPU)
- GPU computing with OpenACC
- **Rotor simulation code Freewake**
- OpenACC port of Freewake
- Conclusion

Freewake: Overview



- Developed 1994-1996 by FT-HS
  - implemented in Fortran
  - MPI-parallel
- Used by the FT-HS rotor simulation code S4
- Simulates the flow around a helicopter's rotor
- Vortex-Lattice method
  - Discretizes complex wake structures with a set of vortex elements
- Based on experimental data (from the international HART program 1995)

Freewake: Comparison with „classical CFD“

„Classical“ CFD:

- Navier-Stokes-equations

Vortex methods:

- Vorticity equation (curl of velocity)

Spatial discretization:

- velocity
- mesh in whole 3D domain

- vorticity
- points/grid in interesting regions

Discretization in time:

- Update velocity using small stencils

- Move points using induced velocity

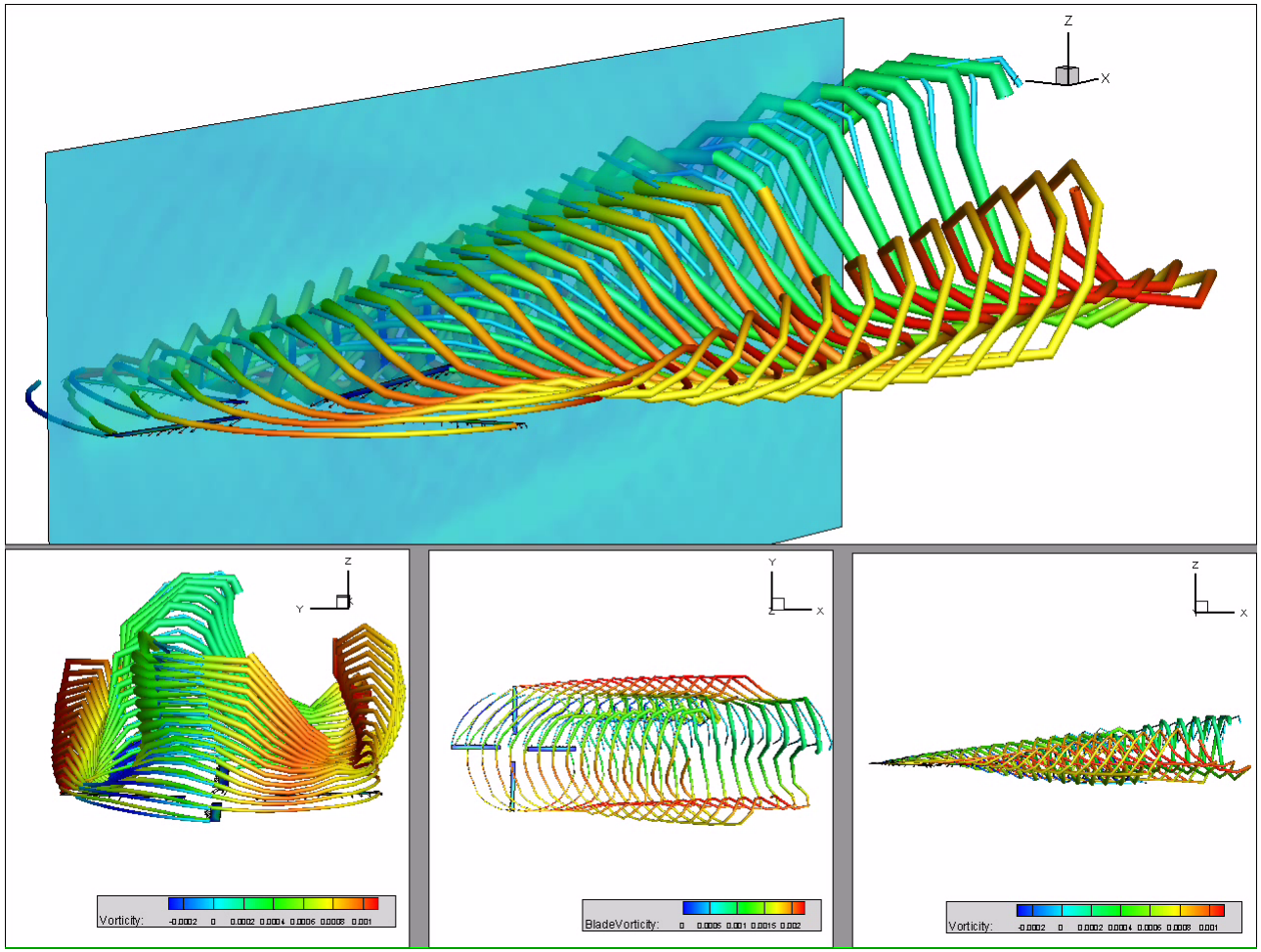
- numerical diffusion

- complex induced velocity calculation
  (similar: N-body problem)

Freewake: Velocity calculation

- moving 2D grid in space with cells of precalculated vorticity
- good initial wake geometry from analytical model
- Velocity at a point:
  - sum over induced velocities of all grid cells
- Choose different formula depending on the distance point ↔ cell:
  - Near range: interpolation to finer grid
  - Medium range: simple linear approach
  - Far range: neglected
- Some cells are interpolated to allow smaller time steps than grid resolution

Freewake: Vortex visualization



Outline

- Hardware-Architecture (CPU+GPU)
- GPU computing with OpenACC
- Rotor simulation code Freewake
- **OpenACC port of Freewake**
- Conclusion

Freewake OpenACC port: Simple benchmark

- Idea: only use formula for medium range

- Performance is compute bound:
  - Working set: $n = \sim 6300$ grid points → $\sim 250\ kB$
    (4 blades with 11x144 points)
  - Operations: $\sim 50 n^2\ Flop$ per iteration

- Fastest CPU implementation (GCC):
  - uses vector-reductions from OpenMP 3.0
  - $\sim 280\ GFlop/s$ (SP)
  - ca. 40x faster than Free-Wake

- OpenACC GPU implementation (PGI):
  - only scalar reductions possible: → not optimal & more complex
  - $\sim 360\ GFlop/s$ (SP)

Freewake OpenACC port: Avoiding branches in loops

if-statements in loops are problematic:

- GPU warp needs several passes for different branches (SIMT)
- may also prevent efficient vectorization on CPUs (SIMD)

Nested if-statements in Free-Wake:

- Grid boundaries: (and 1/8 turn behind blade)
  → partly unrolled loops by hand to handle all cases individually

- "Flags":
  → Calculate $result = flag * a + (1 - flag) * b$ for $flag \in \{0,1\}$

- Different formulas depending on distance point ↔ cell:
  - difficult!
  - currently best results with dedicated loops for individual cases

Freewake OpenACC port: Hybrid CPU/GPU calculations

- MPI-parallel:
  - Grid stored redundantly on all processes
  - Each process calculates the velocity of a set of points

- Hybrid calculation:
  - First MPI-process uses the GPU
  - All others stay on the CPU
  - uses `acc_set_device_type( acc_device_... )`

→ We need load balancing!

Freewake OpenACC port: Load balancing

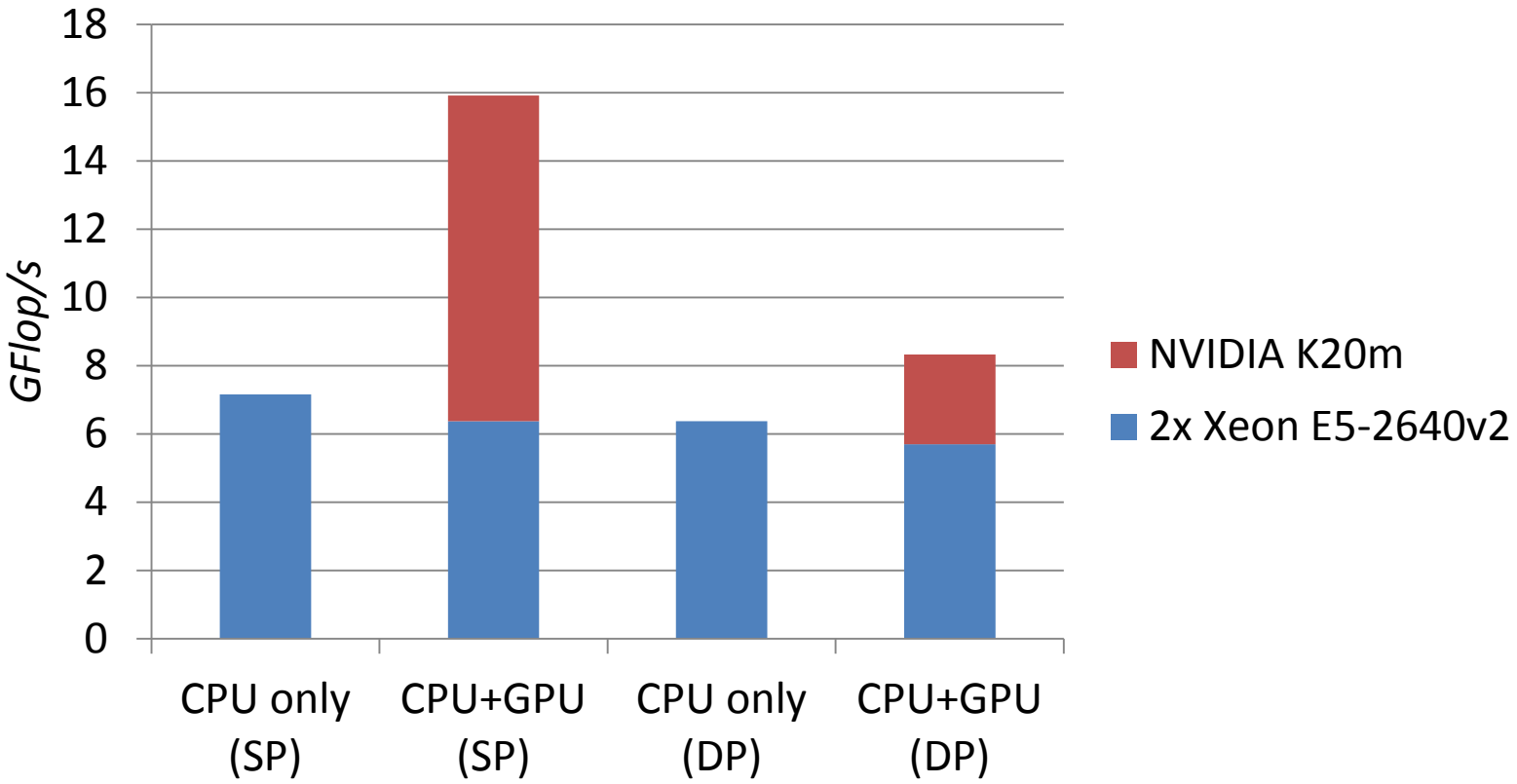
Problems:

- Computational costs of grid points vary
- different performance of CPU core and GPU

→ **Dynamic load balancing:**

- Measure required runtime in each iteration
- Estimate work of each element with $time/n\_elems$
- Redistribute the work equally among all processes
  - → algorithm unaware of CPU and GPU performance
  - → still provides reasonable distributions after some iterations

Freewake OpenACC port: performance results



Outline

- Hardware-Architecture (CPU+GPU)
- GPU computing with OpenACC
- Rotor simulation code Freewake
- OpenACC port of Freewake
- **Conclusion**

Conclusion

- Successfully ported the Freewake simulation to GPUs using OpenACC
  - original numerical method not modified
  - refactored & restructured a lot of code
  - results verified on CPU and GPU

→ Porting complex algorithms to GPUs is difficult
  - branches in loops hurt (much more than for CPUs)

- Loop restructuring may also improve the CPU performance
  (SIMD vectorization on modern CPUs)

- Stumbled upon several OpenACC PGI-compiler bugs (all fixed very fast)

Future work:

- Performance optimization for GPUs and CPUs:
  - also consider numerical modifications
  - improve data layout

- New features:
  - individual blade movement
  - extension to wind turbines
  - fuselage-rotor interference
  - multiple interacting rotors
  - variable RPM

- Algorithmic improvements:
  - refined convergence criterion
  - calculate induced velocity on tree structures:
    →reduce computational costs from $O(n^2)$ to $O(n\,log(n))$

Questions?