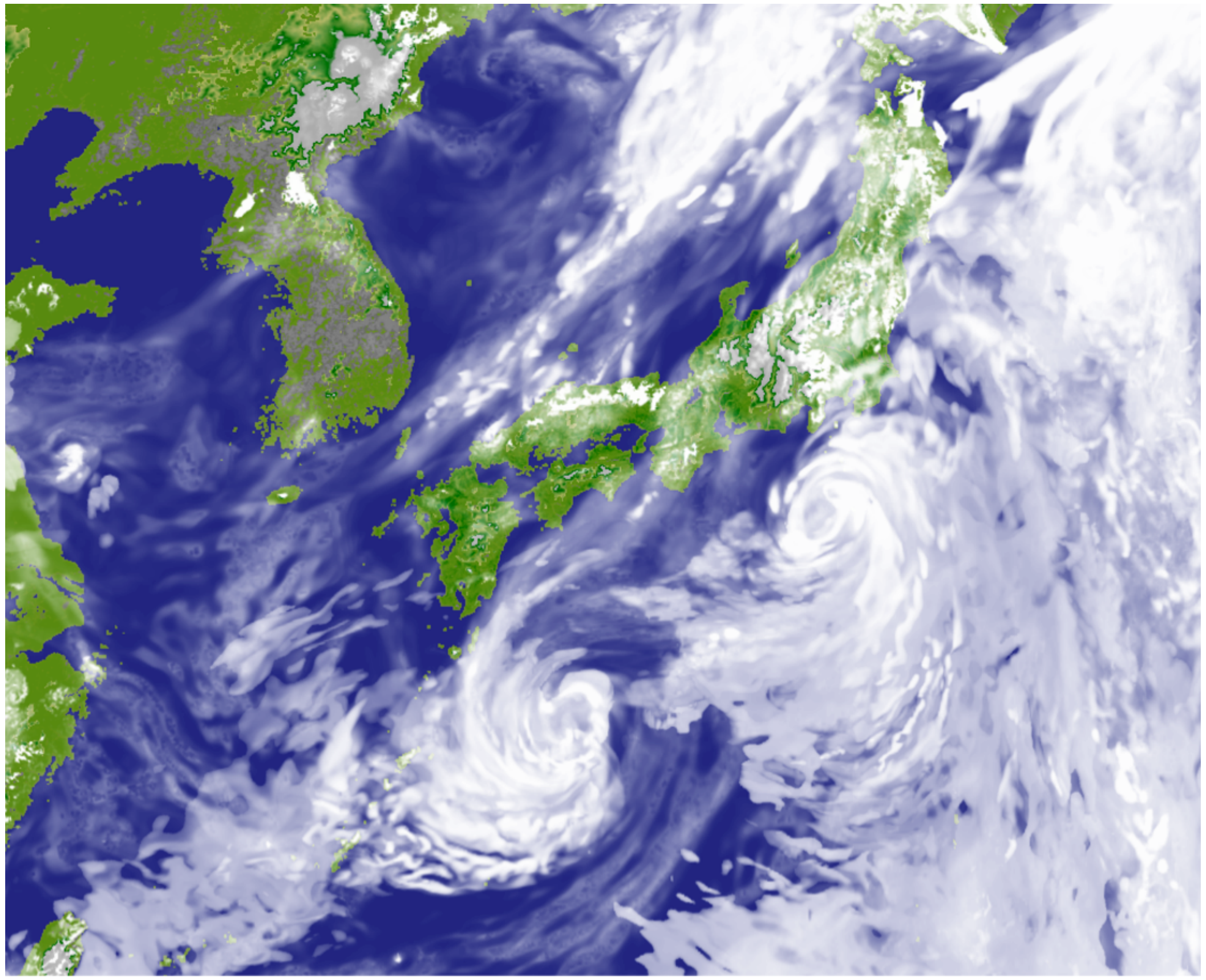
Hybrid Fortran
High Performance & Productivity for GPU Numerics

Talk



Michel Müller
Postdoctoral Researcher ETH Zurich
Dr. Eng., Tokyo Institute of Technology
MSc. ETH in Electrical Engineering and Information Technology

2018-09-27

Outline

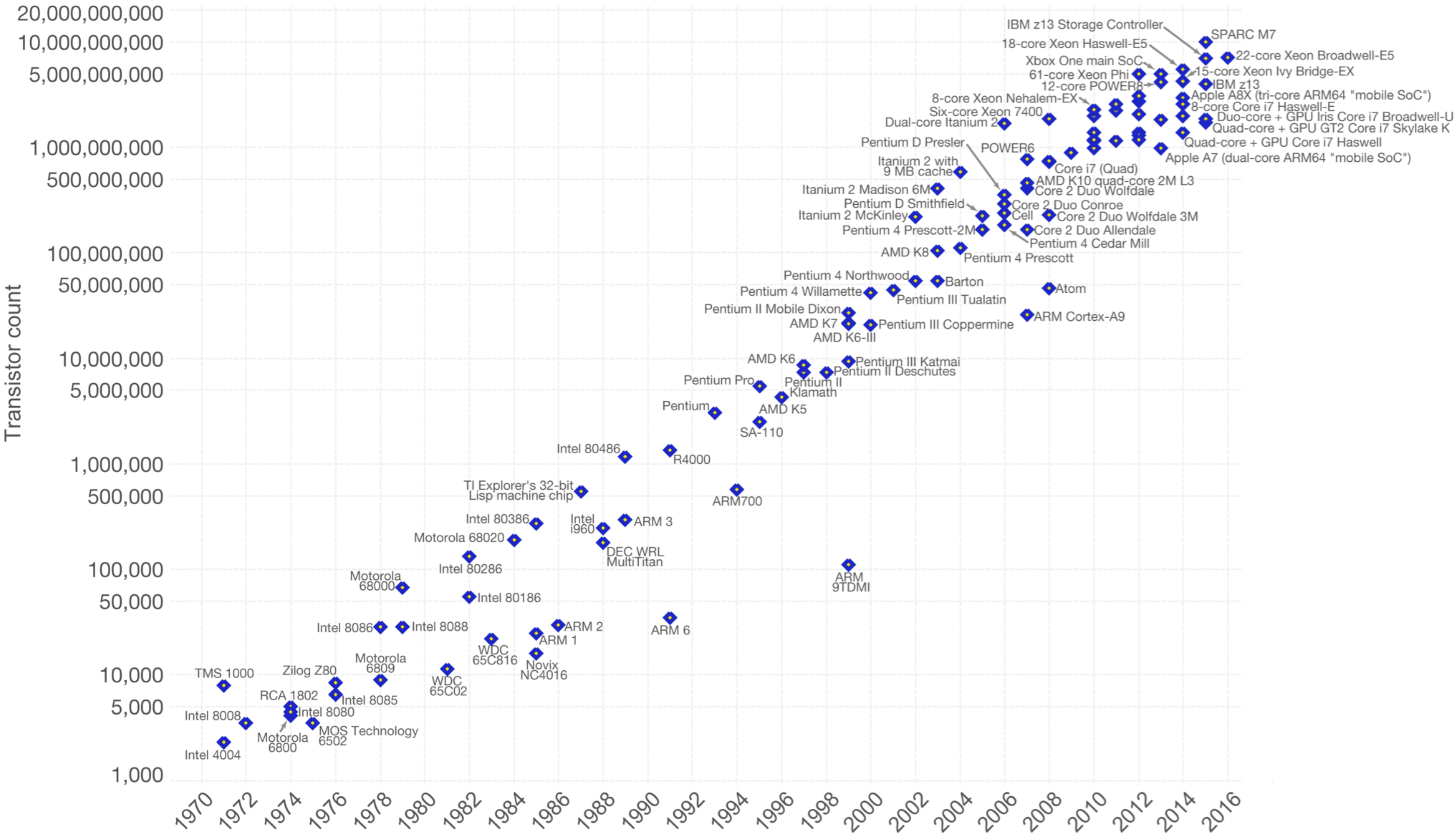
1. Introduction
2. Method
3. Application
4. Performance
5. Method Comparison
6. Conclusion

NWP and Computational Performance

- Increase in computational performance allows **increasing grid resolution**.
  - During last decade this allows **resolution of** increasingly **small cloud formations** in dynamical core.
- Typically applied finite-volume and finite-difference based discretization methods are **bottlenecked by memory bandwidth** in the dynamics.

➡ Hardware architectures with high memory bandwidth are seeked.

Moore's Law still holds, …



Data Source: https://en.wikipedia.org/wiki/Transistor_count,
Visualization by Max Roser (CC-BY-SA)

… however Dennard scaling does not.

Dennard scaling: Power density of micro transistors proportional to area.

➡ Clock frequency/single threaded perf. scales inverse proportionally to transistor size

Since 90nm process technology (~2004-2005), Dennard scaling **does not hold anymore**.

Leakage currents increasingly **limit advancements in single threaded performance**.

Latency- versus Throughput Oriented Processing

Latency: Time elapsed between initiation and completion of a task.

Throughput: Total amount of work completed per unit time.

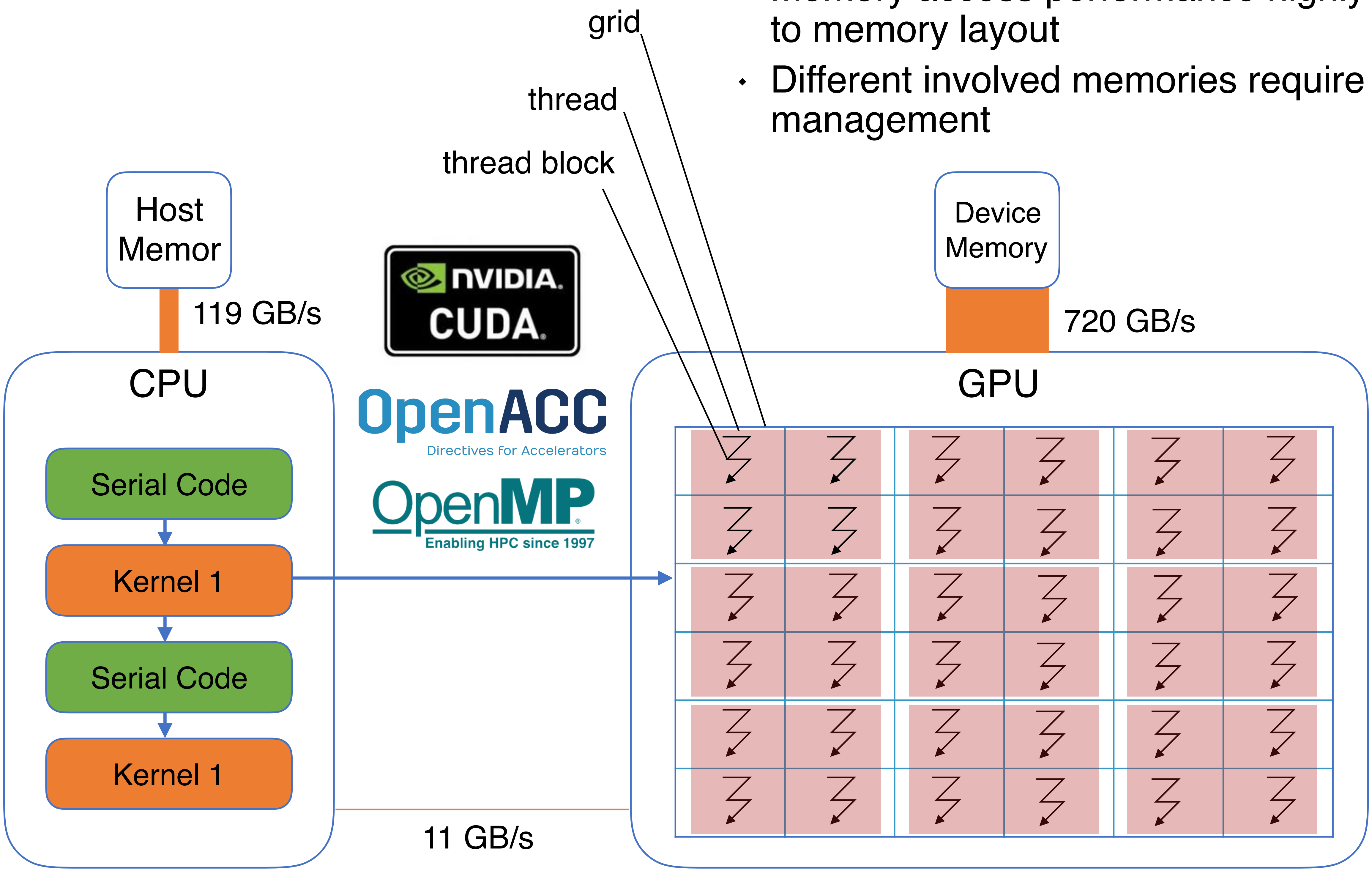
- Due to end of Dennard scaling:
  - ➡ shift from latency-oriented processor design to throughput-oriented
  - ➡ applications only profit when adapted accordingly

GPU Computing

- Graphics Processing Units (**GPUs**) are a **popular type of throughput-oriented processors**.
- Today has many applications outside of graphics.
- **Applications need to be highly parallelizeable**, as GPUs have a **high latency** to complete a single task compared to CPUs.

GPU Computing

- High memory bandwidth
- Support for branching, 64bit FP
- Fine-grained parallelism
- Memory access performance highly sensitive to memory layout
- Different involved memories require management



ASUCA NWP Model

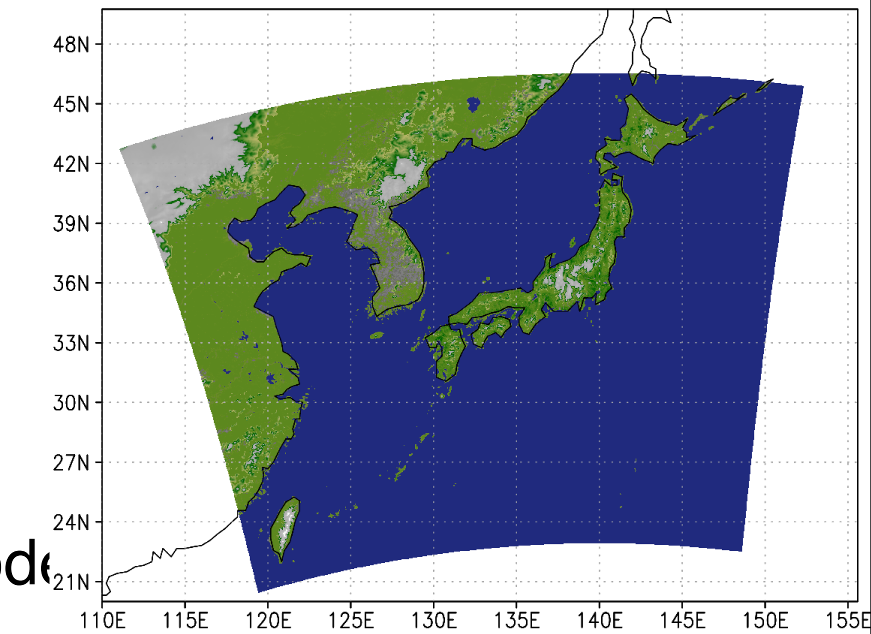


Figure 1.2: ASUCA's model simulation boundaries

What is ASUCA**?**

- ``**A**suca is a **S**ystem based on a **U**nified **C**oncept for **A**tmosphere"
- **fully compressible**, **non-hydrostatic** weather prediction mode
- **regional scale** - as depicted in Figure 1.2
- one of **main operational** forecast models in Japan, **in production** since 2014
- spatial discretization: **finite-volume method** on **Arakawa-C-type rectangular** grid
- time discretization:
  - **third-order Runge-Kutta** based iteration scheme for **advection and Coriolis** force
  - **time-splitting method**, employing secondary third-order Runge-Kutta iteration with **short time step** for **sound-** and **gravity waves**
- **vertical-only** models for parametrization of **radiation**, **planetary boundary layer** and **surface physical processes**

GPUs for Numerical Weather Prediction

- **GPUs** offer **high memory bandwidth**, which is in high demand in NWP.

  ➡ GPUs are an **attractive target** architecture.
- Major **problems** to solve for existing regular grid NWP codes:
  - **Memory layout** needs change
  - **Code granularity** in physical processes too coarse for GPU
- **Existing methods** to solve these problems:
  - Only apply GPU to dynamical core.
  - Rewrite Fortran code using C++ templates for architecture specialization.
  - Code divergence between CPU and GPU to solve granularity issues.

➡ **Unsatisfactory** to maintain a unified, coherent and efficient code base in Fortran (the standard in NWP)

➡ **For ASUCA, a solution with none of these drawbacks was sought.**

Background

- ✓ paradigm shift towards throughput oriented design
- ✓ GPUs attractive for NWP (high mem. bandwidth)
- ✓ productivity and maintainability of GPU approaches lacking

Motivation

- ✓ Many of today's NWP- and climate models cannot make efficient use of high-throughput architectures. We want to find and prove easily adoptable approach.

Goal

- ✓ GPU port for "ASUCA" NWP model in Fortran with minimal code divergence / minimal learning

Contributions

- ☐ **new granularity abstraction and memory layout transformation method**
- ☐ applied to ASUCA, resulting in >3x speedup in kernel performance and >2x reduction in processors required for a full scale run with real data
- ☐ method unique in increasing productivity for porting coarse-grained codes to GPU

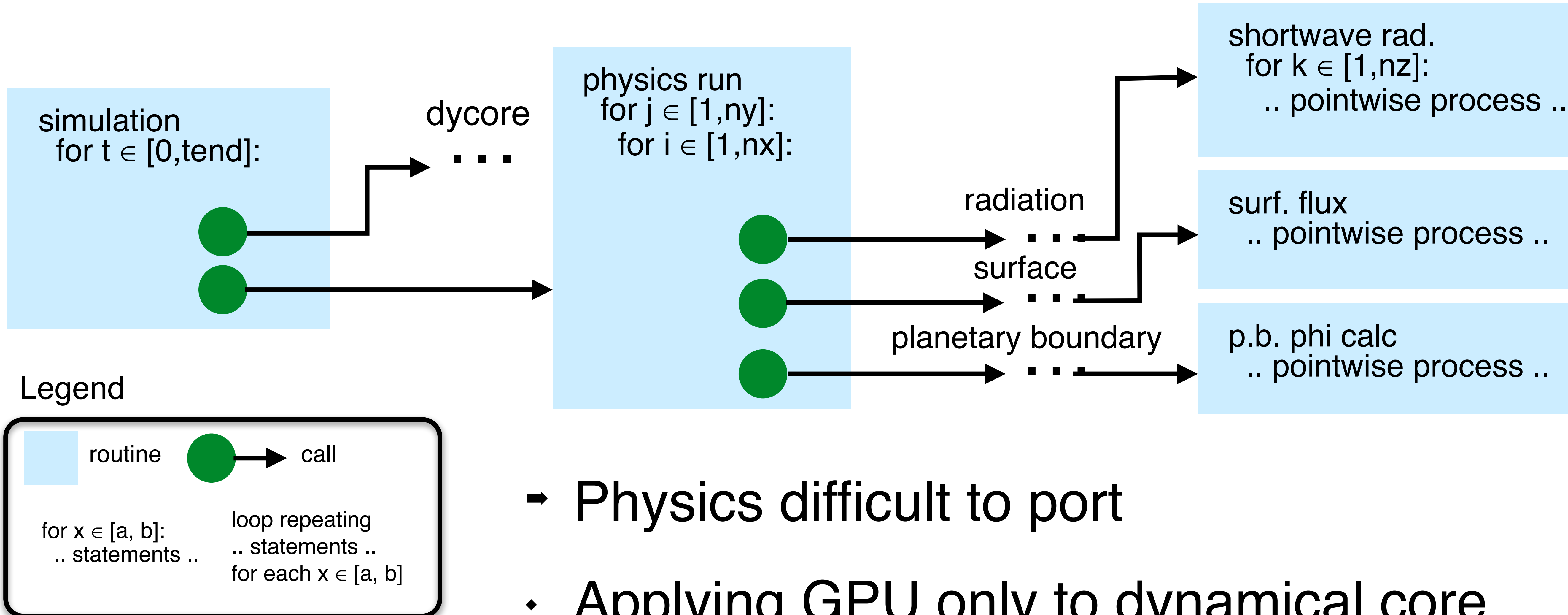
2. Method

- Granularity Abstraction
- Memory Layout & Regions
- Code Transformation

Assumptions for Design

- **Mainly used data structure** is **Fortran arrays** of different dimensions and data types.
- Kernels are **data parallel**.
- **Existing** inter-node / inter-GPU **communication** code can be **reused**.

ASUCA Code Structure



- Physics difficult to port
- Applying GPU only to dynamical core requires expensive host-device-communication for every timestep

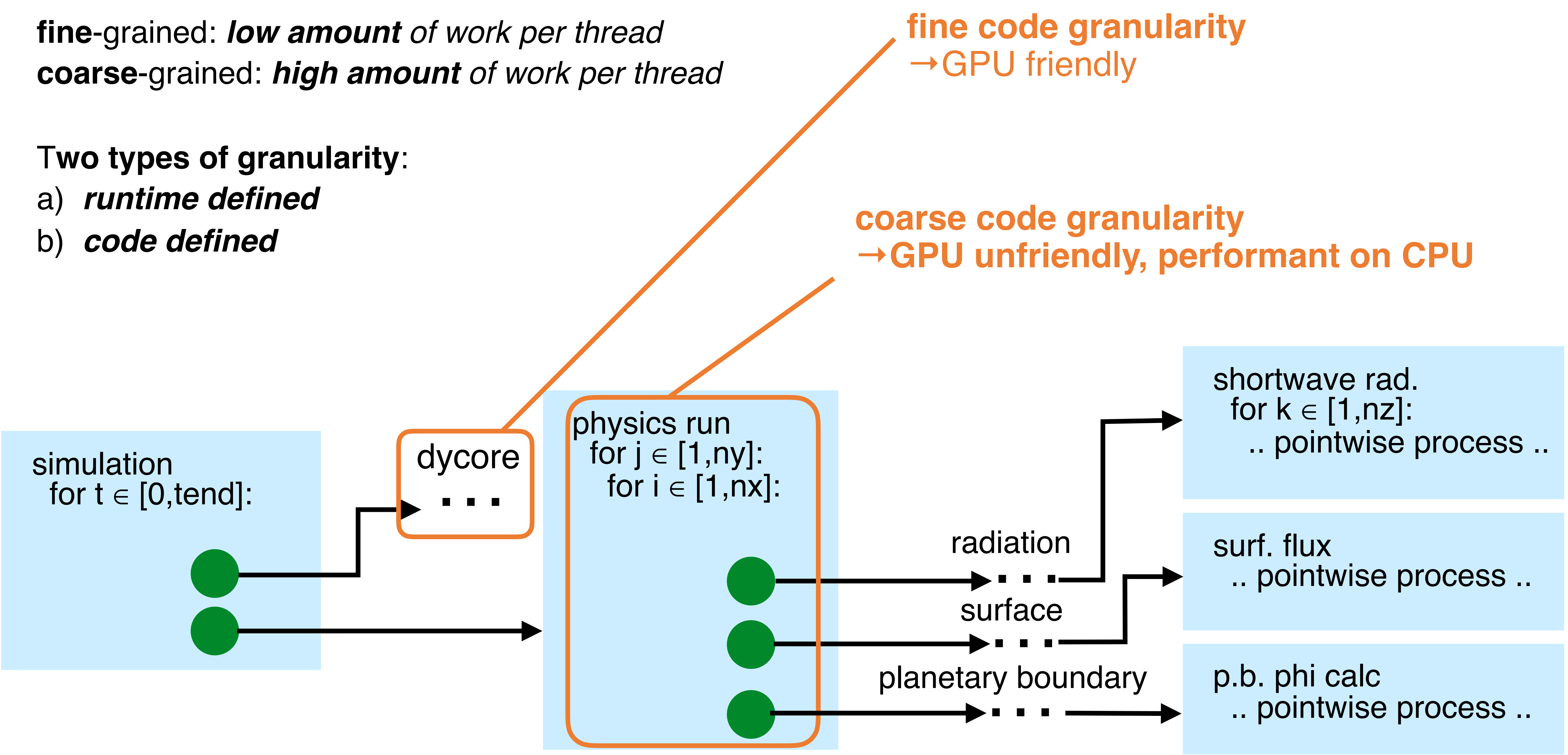
Key Problems

1. Code Granularity

Definition of **granularity**:
*The **amount of work** done **by one thread**.*

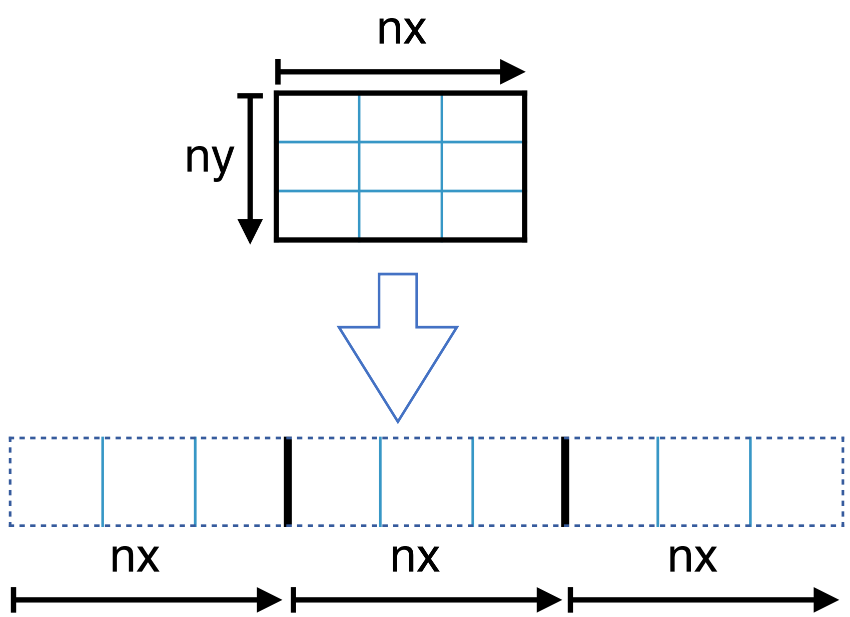
fine-grained: ***low amount** of work per thread*
coarse-grained: ***high amount** of work per thread*

Two types of granularity:
a) ***runtime defined***
b) ***code defined***

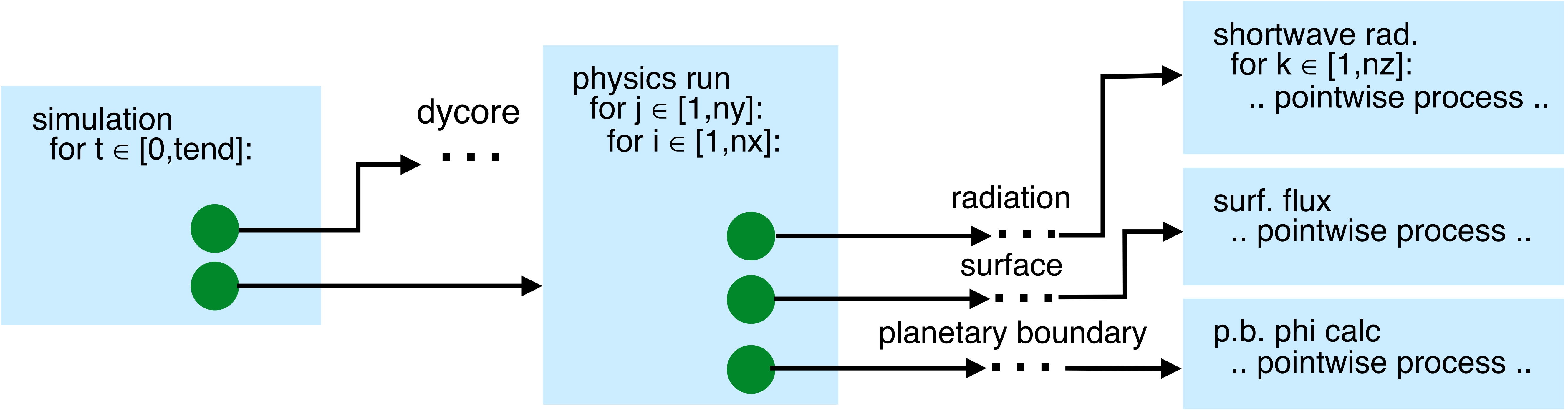


Key Problems

2. Memory Layout



- **Performant** layout **on CPU**: Keep **fast varying** vertical domain **in cache → k-first** Example stencil in original code: A_out(k,i,j) = A(k,i,j) + A(k,i-1,j) …
- **GPU**: Requires **i-first or j-first** for **coalesced access**



Hybrid Fortran

Main ideas:

- Allow **efficient** many-core **GPU** port while **maintaining multi-core CPU** compatibility
- Delegate parallelization boilerplate to framework
- Allow **multiple parallelization granularities** for the same code
- **Transform memory layout** for each target architecture

Parallelization & Granularity Abstraction

```
do i = 1, nx
  do j = 1, ny
    ! ..pointwise code..
```

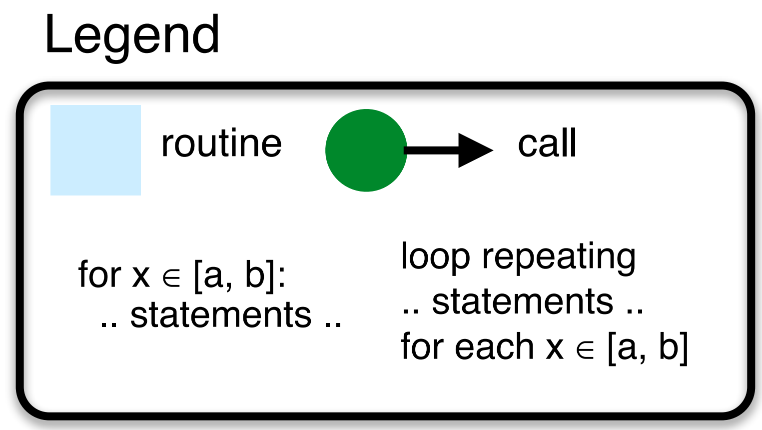
```
@parallelRegion{
  domName(i,j), domSize(nx,ny), appliesTo(CPU)
}
! ..pointwise code..
```

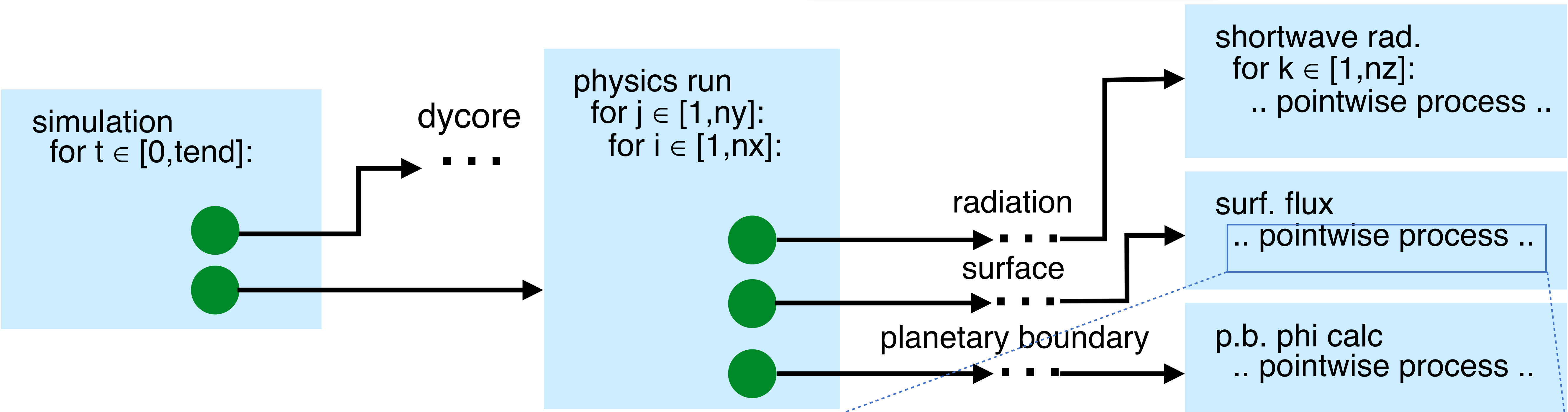
explicit parallelization - orthogonal to sequential loops

allows multiple parallelization granularities

Creates CUDA Fortran, OpenACC or CPU multicore-OpenMP based parallelization, depending on backend.

Example Physical Process





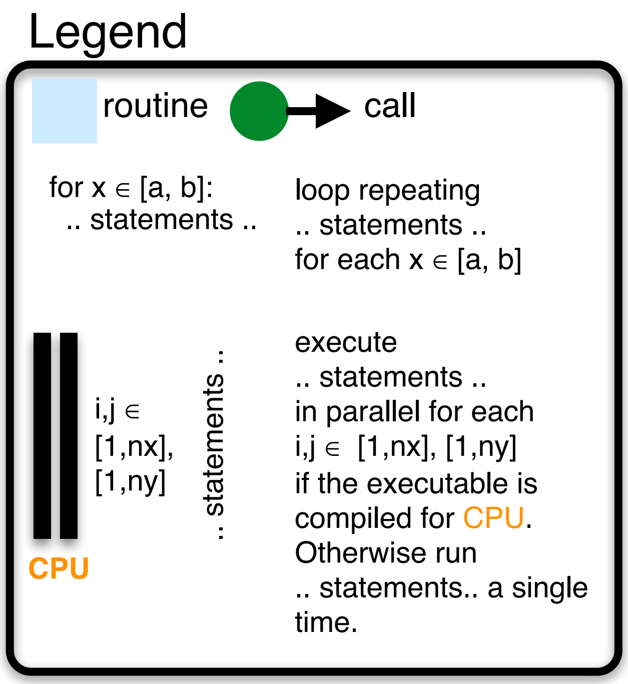
example reference code from surface flux

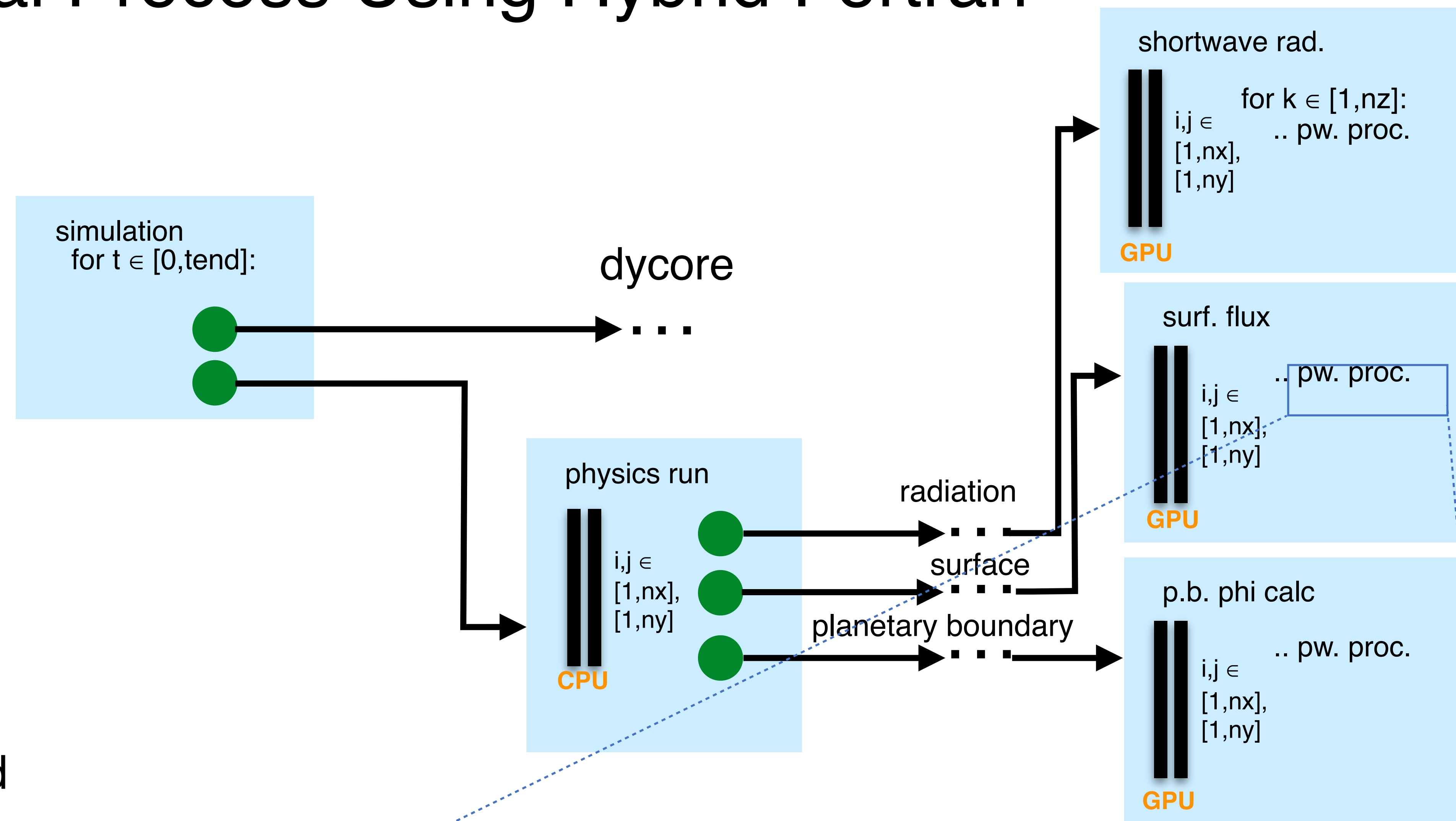
- data parallelism not exposed at this layer of code
- ➡ coarse-grained parallelization

```
lt = tile_land
if (tlcvr(lt) > 0.0_r_size) then
  call sf_slab_flx_land_run(&
    ! ... inputs and further tile variables omitted
    & taux_tile_ex(lt), tauy_tile_ex(lt) &
    & )

  u_f(lt) = sqrt(sqrt(taux_tile_ex(lt) ** 2 + tauy_tile_ex(lt) ** 2))
else
  taux_tile_ex(lt) = 0.0_r_size
  tauy_tile_ex(lt) = 0.0_r_size
  ! ... further tile variables omitted
end if
```

Example Physical Process Using Hybrid Fortran





- pointwise code reused
- code transformed to apply fine-grained parallelism
  - appliesTo clause to specify parallelization target
  - call graph transformed globally to expose data parallelism at required granularity

```
@parallelRegion{appliesTo(GPU), domName(i,j), domSize(nx,ny)}
lt = tile_land
if (tlcvr(lt) > 0.0_r_size) then
  call sf_slab_flx_land_run(&
    ! ... inputs and further tile variables omitted
    & taux_tile_ex(lt), tauy_tile_ex(lt) &
    & )

  u_f(lt) = sqrt(sqrt(taux_tile_ex(lt) ** 2 + tauy_tile_ex(lt) ** 2))
else
  taux_tile_ex(lt) = 0.0_r_size
  tauy_tile_ex(lt) = 0.0_r_size
  ! ... further tile variables omitted
end if
! ... sea tiles code and variable summing omitted
@end parallelRegion
```

2. Method

Data Specifications

Data specifications:

- autoDom: extend existing data domain specification with parallel domain given by @domainDependant directive.
  - domName, domSize attributes specify horizontal extension of data domain
- present: data is already present on device.
  - requires counterpart specification at data region boundaries with transferHere attribute

```
@domainDependant{domName(i,j), domSize(nx,ny), attribute(autoDom, present)}
tlcvr, taux_tile_ex, tauy_tile_ex, u_f
@end domainDependant

@parallelRegion{appliesTo(GPU), domName(i,j), domSize(nx,ny)}
lt = tile_land
if (tlcvr(lt) > 0.0_r_size) then
  call sf_slab_flx_land_run(&
    ! ... inputs and further tile variables omitted
    & taux_tile_ex(lt), tauy_tile_ex(lt) &
    & )

  u_f(lt) = sqrt(sqrt(taux_tile_ex(lt) ** 2 + tauy_tile_ex(lt) ** 2))
else
  taux_tile_ex(lt) = 0.0_r_size
  tauy_tile_ex(lt) = 0.0_r_size
  ! ... further tile variables omitted
end if
! ... sea tiles code and variable summing omitted
@end parallelRegion
```

Transformed Code

Example surface flux kernel transformed with OpenACC backend.

OpenACC parallelization

data specifications

block size macros

parallel loops

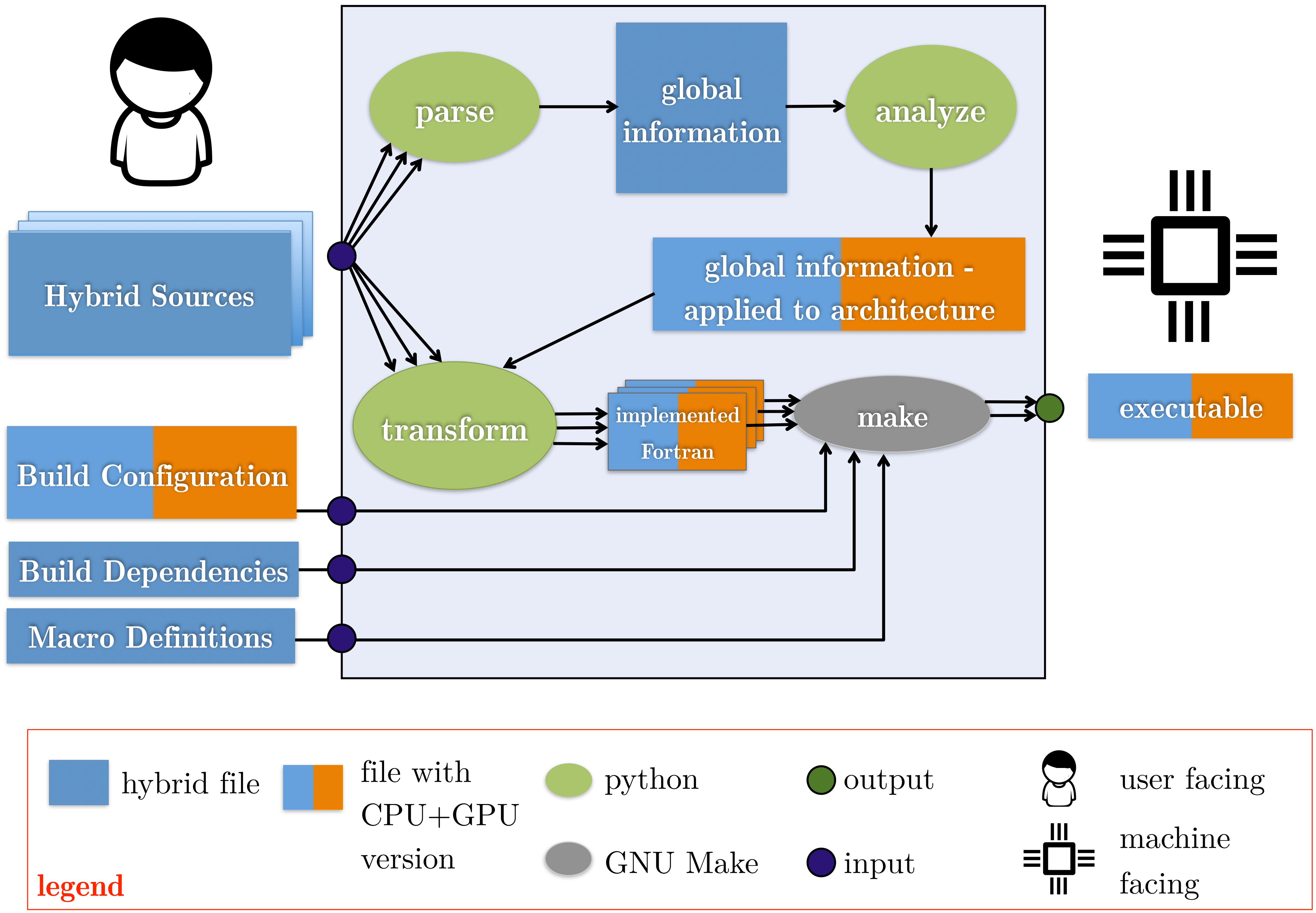
horizontal domain extension of data

storage ordering macro

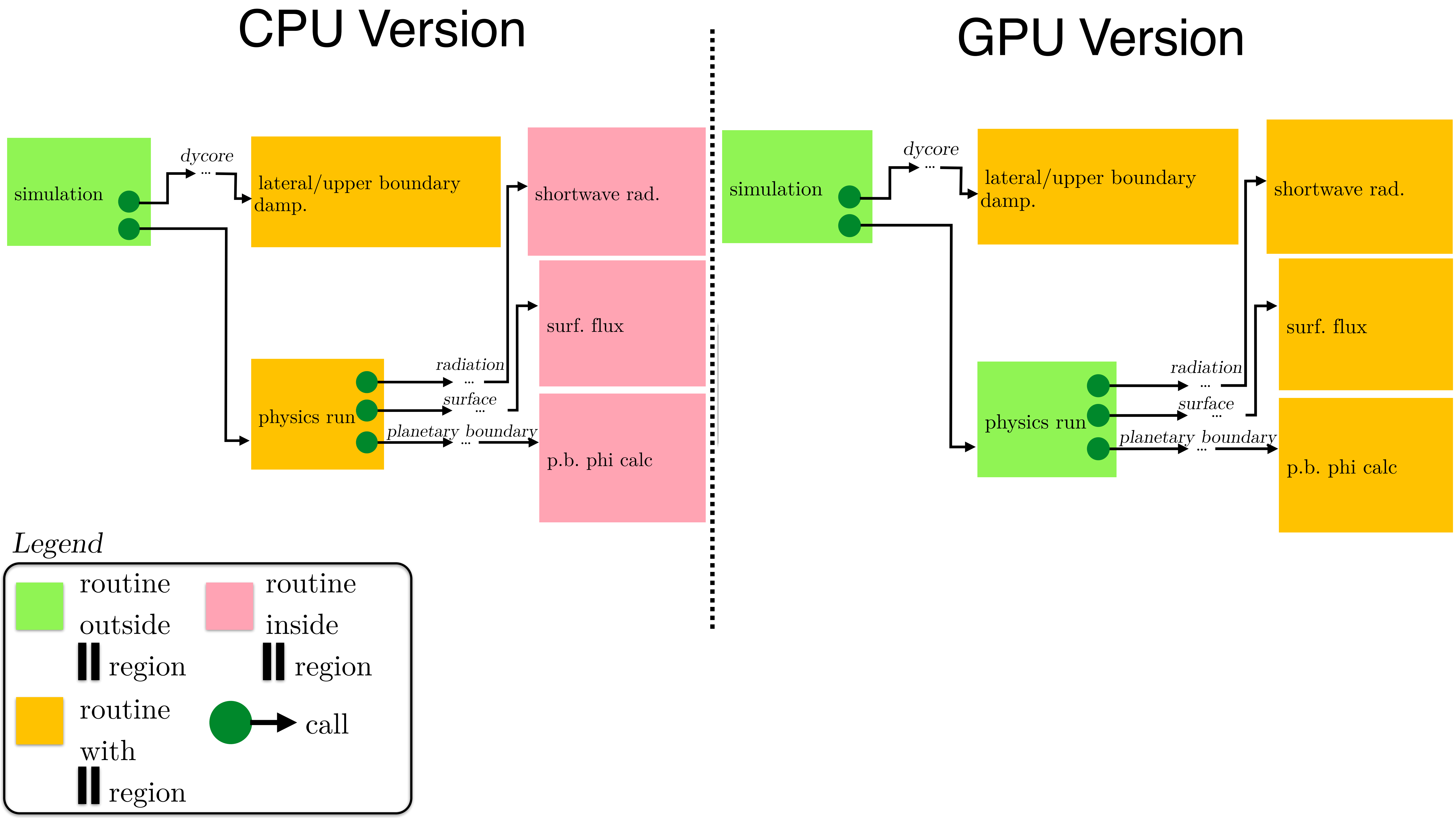
line breaks

```
!$acc kernels deviceptr(taux_tile_ex) deviceptr(tauy_tile_ex)  &
!$acc& deviceptr(tlcvr) deviceptr(u_f)
!$acc loop independent vector(CUDA_BLOCKSIZE_Y)
outerParallelLoop0: do j=1,ny
!$acc loop independent vector(CUDA_BLOCKSIZE_X)
  do i=1,nx
      ! *** loop body *** :
    lt = tile_land
    if (tlcvr( AT(i,j,lt) )> 0.0_r_size) then
            call sf_slab_flx_land_run(&
                ! ... inputs and further tile variables omitted
                & taux_tile_ex( AT(i,j,lt) ), tauy_tile_ex( AT(i,j,lt) )
                    ↪  &
                & )
      u_f( AT(i,j,lt) )= sqrt(sqrt(taux_tile_ex( AT(i,j,lt) )** 2 + &
          & tauy_tile_ex( AT(i,j,lt) )** 2))
    else
      taux_tile_ex( AT(i,j,lt) )= 0.0_r_size
      tauy_tile_ex( AT(i,j,lt) )= 0.0_r_size
      ! ... further tile variables omitted
    end if
    ! ... sea tiles code and variable summing omitted
  end do
end do outerParallelLoop0
!$acc end kernels
```

Transformation Process



Callgraph Analysis



Limitations

- code for **programmable caches** on GPU (“shared memory”, “texture memory”) is **not generated** by tool.
- relies on standard **subroutines**, e.g. Fortran **function** construct not supported for code running on GPU.

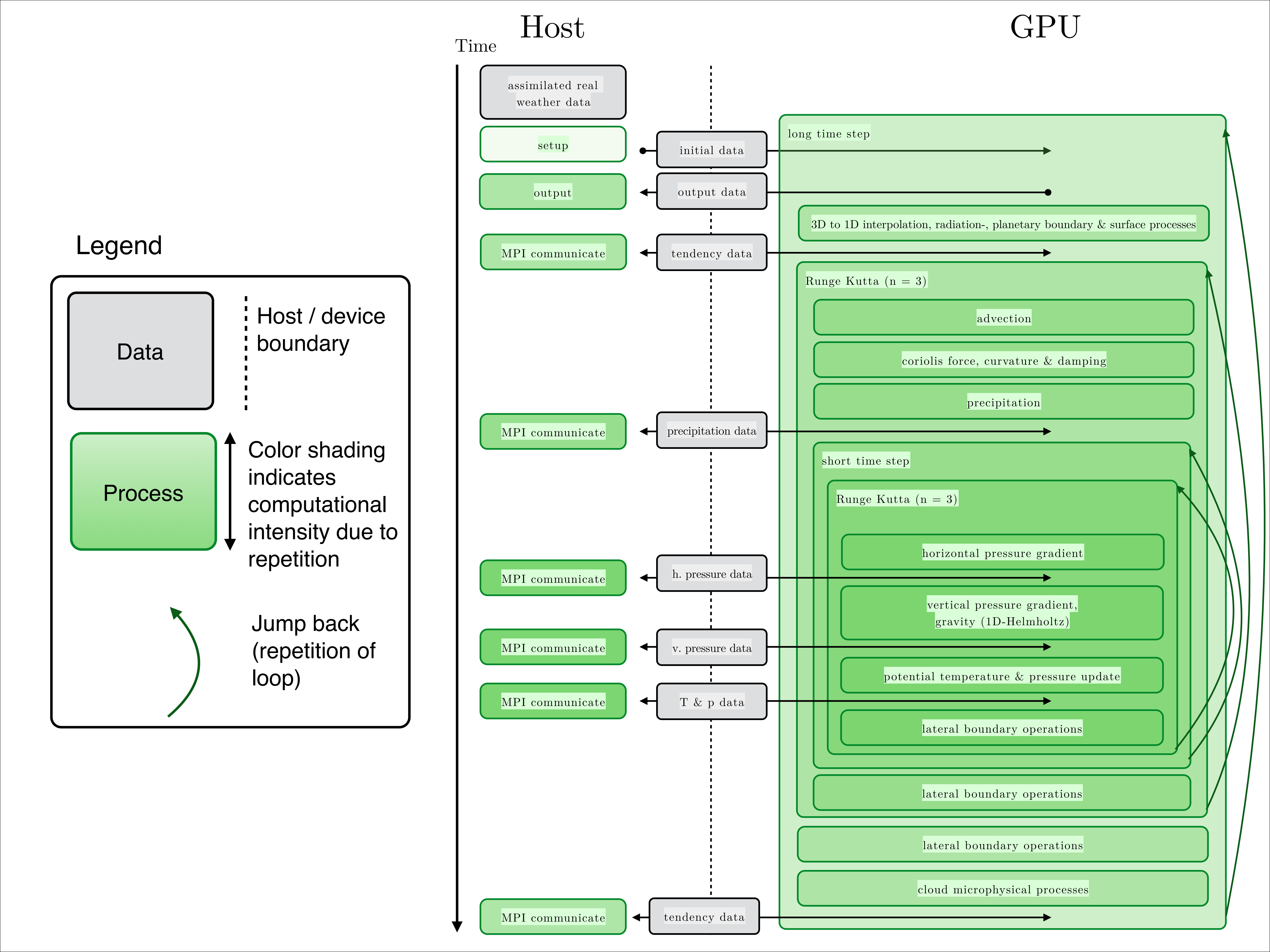
Contributions

- ✓ new granularity abstraction and memory layout transformation method
- ☐ **applied to ASUCA**, resulting in >3x speedup in kernel performance and >2x reduction in processors required for a full scale run with real data
- ☐ method unique in increasing productivity for porting coarse-grained codes to GPU

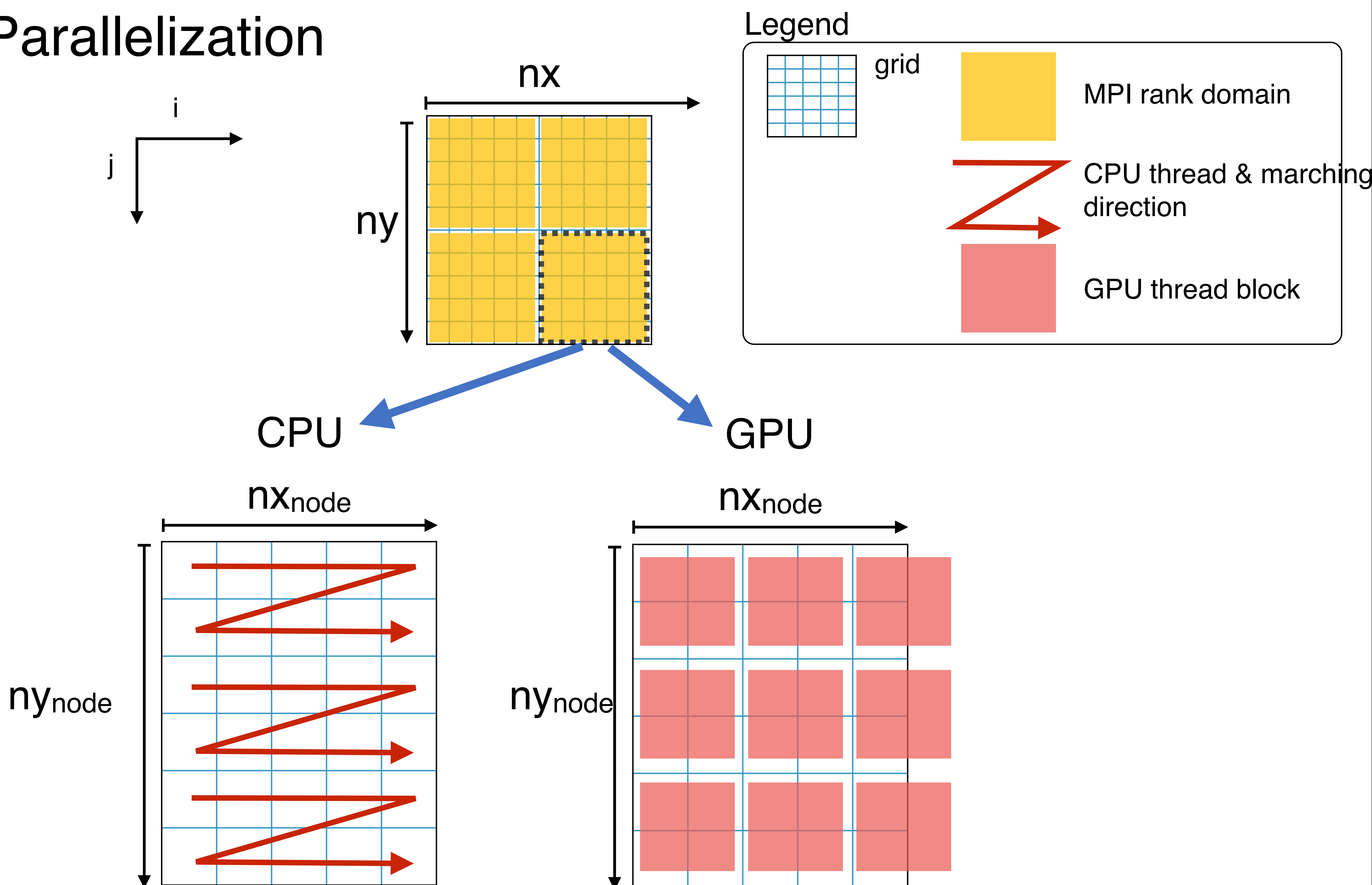
3. Application

- Hybrid ASUCA Implementation
- Productivity Results

Introduction → Method → Application → Performance → Method Comparison → Conclusion


Host
GPU
Time
assimilated real weather data
setup
initial data
long time step
output
output data
3D to 1D interpolation, radiation-, planetary boundary & surface processes
MPI communicate
tendency data
Runge Kutta (n = 3)
advection
coriolis force, curvature & damping
precipitation
MPI communicate
precipitation data
short time step
Runge Kutta (n = 3)
horizontal pressure gradient
MPI communicate
h. pressure data
vertical pressure gradient, gravity (1D-Helmholtz)
MPI communicate
v. pressure data
potential temperature & pressure update
MPI communicate
T & p data
lateral boundary operations
lateral boundary operations
lateral boundary operations
cloud microphysical processes
MPI communicate
tendency data
Legend
Data
Host / device boundary
Process
Color shading indicates computational intensity due to repetition
Jump back (repetition of loop)

Parallelization

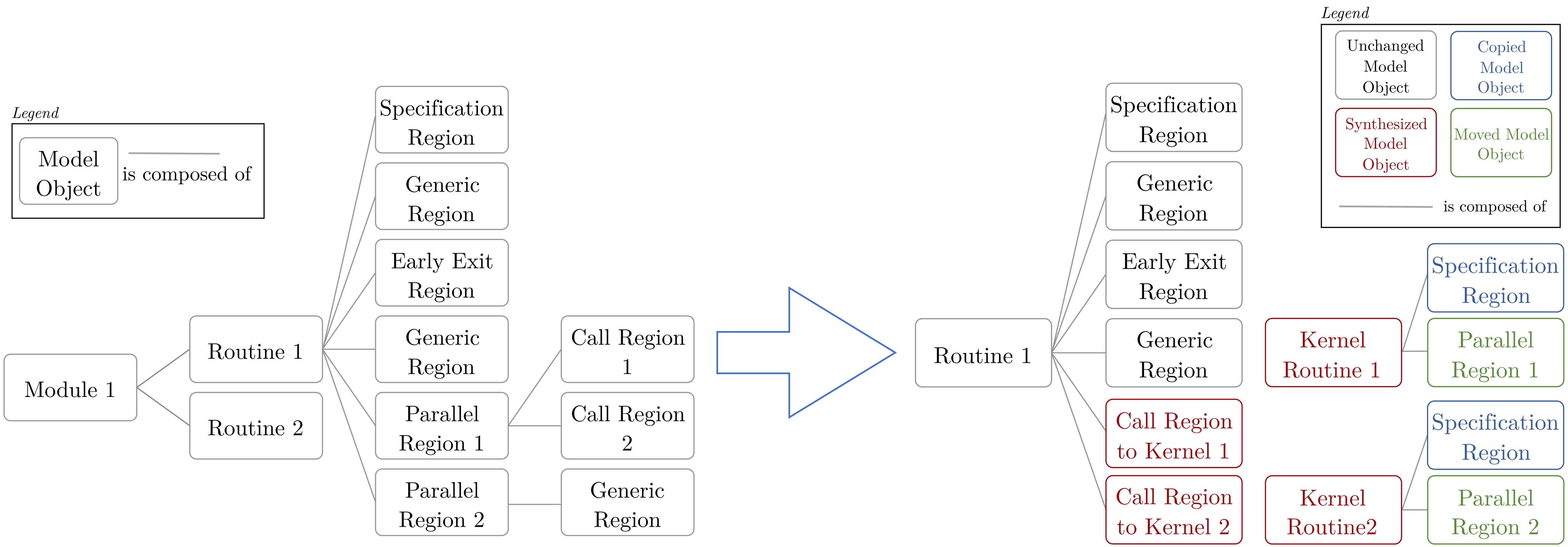


Dynamical Core

- ASUCA's dynamical core contains many "tight parallel loops", i.e. fine-grained parallelism.
- CUDA Fortran compiler was most stable during development.
  - ➡ Chosen as main backend.
  - ➡ Transformed code must have separate routines per kernel.

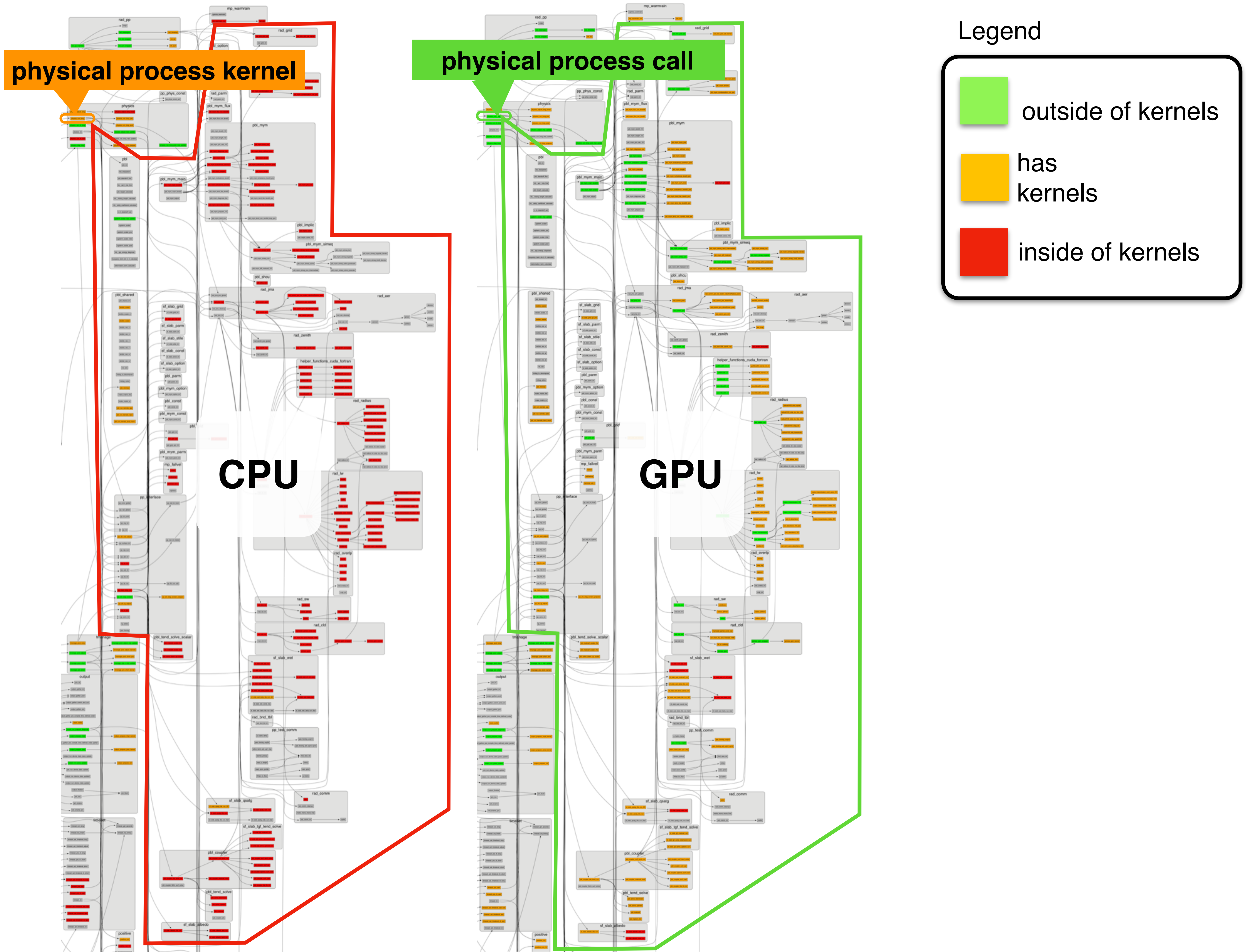
➡ To facilitate tight parallel loops, Hybrid Fortran employs routine splitting.

- ➡ Existing code becomes compatible with CUDA Fortran backend.



Physical Processes

- Original **physical process library** from JMA adapted for GPU (**MSM0705 model**) provides column-wise models for:
  - **Radiation**, (solar, optical cloud absorption, atmospheric reflection and absorption)
    - For efficient use of GPU, memory footprint of indirect radiation effects was reduced by 10x by using ad-hoc computations for each long-wave band rather than storing temporary data of all bands.
  - **Planetary boundary layer model**
  - Wind momentum-, sensible heat- and latent heat **surface fluxes**
- **Kessler-type warm rain** model
- Hybrid Fortran’s **adaptive parallelization granularity** used to generate GPU version


physical process kernel
physical process call
CPU
GPU
Legend
outside of kernels
has kernels
inside of kernels

Column-wise Courant-Friedrichs-Lewy Convergence

- **Precipitation** module uses **separate CFL condition per column**.
- Due to granularity shift, column-wise CFL convergence **requires** change from simple loop break to **reduction kernel** and **masking array** (cfl_reached).

```
@domainDependant{
    attribute(autoDom, present), domName(i,j), domSize(nx,ny)
}
cfl_reached, dt_rk_rest, ...
@end domainDependant

! ... initialization of variables

timestep_sed: do
    ! ... Runge-Kutta based iterative solution to sedimentation

    @parallelRegion{appliesTo(GPU), domName(i, j), domSize(nx, ny)}
    if ( dt_rk_rest < dt_rk_rest_epsln ) then
      cfl_reached = .true.
    end if
    @end parallelRegion

    call all_true_for_xy_plane(cfl_reached, all_cfl_reached)

    if (all_cfl_reached) then
      exit timestep_sed
    end if
end do timestep_sed
```

Verification

- Hybrid ASUCA uses 64bit FP arithmetic throughout.
- Normalized root mean square error was tested continuously for pressure, moment and temperature variables. Stays within 1E-9.
- Performed tests include:
    - Radiation test.
    - Physical process test for radiation, planetary boundary layer and surface.
    - Two-dimensional “warm bubble” test.
    - Various application configurations with real data, including full scale test on 1581x1301x58 grid (2km resolution).

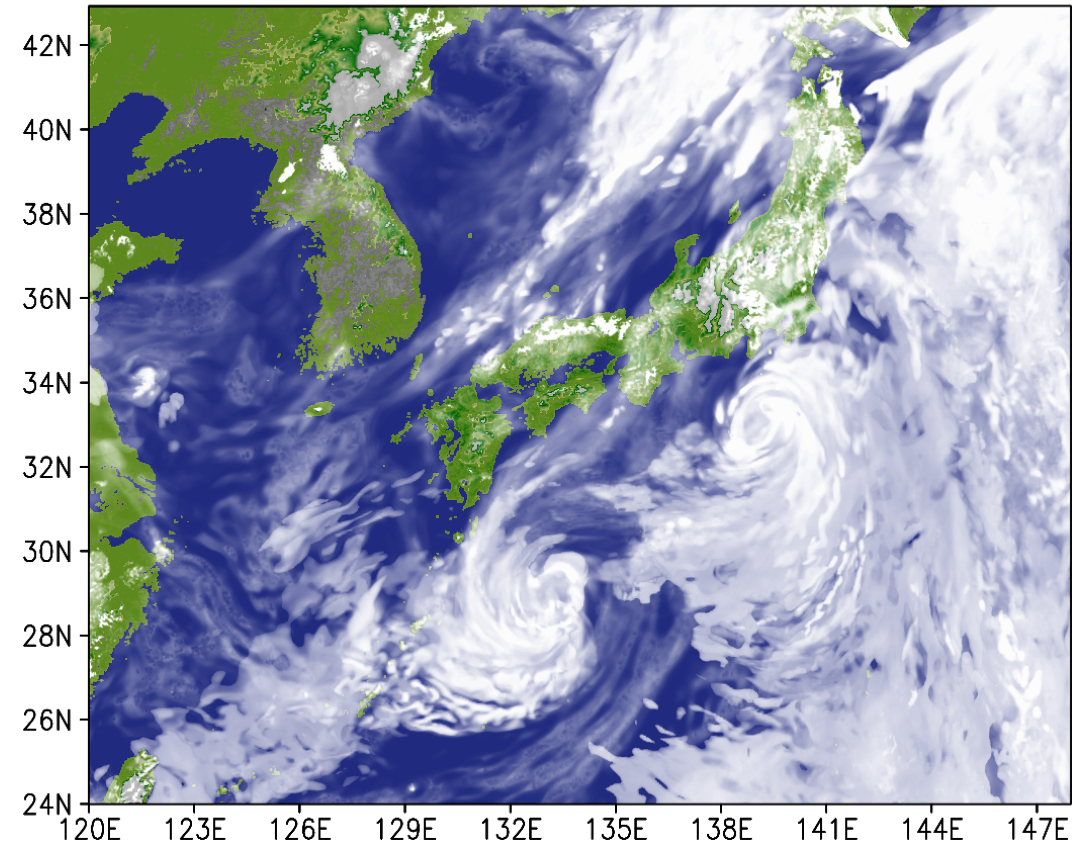
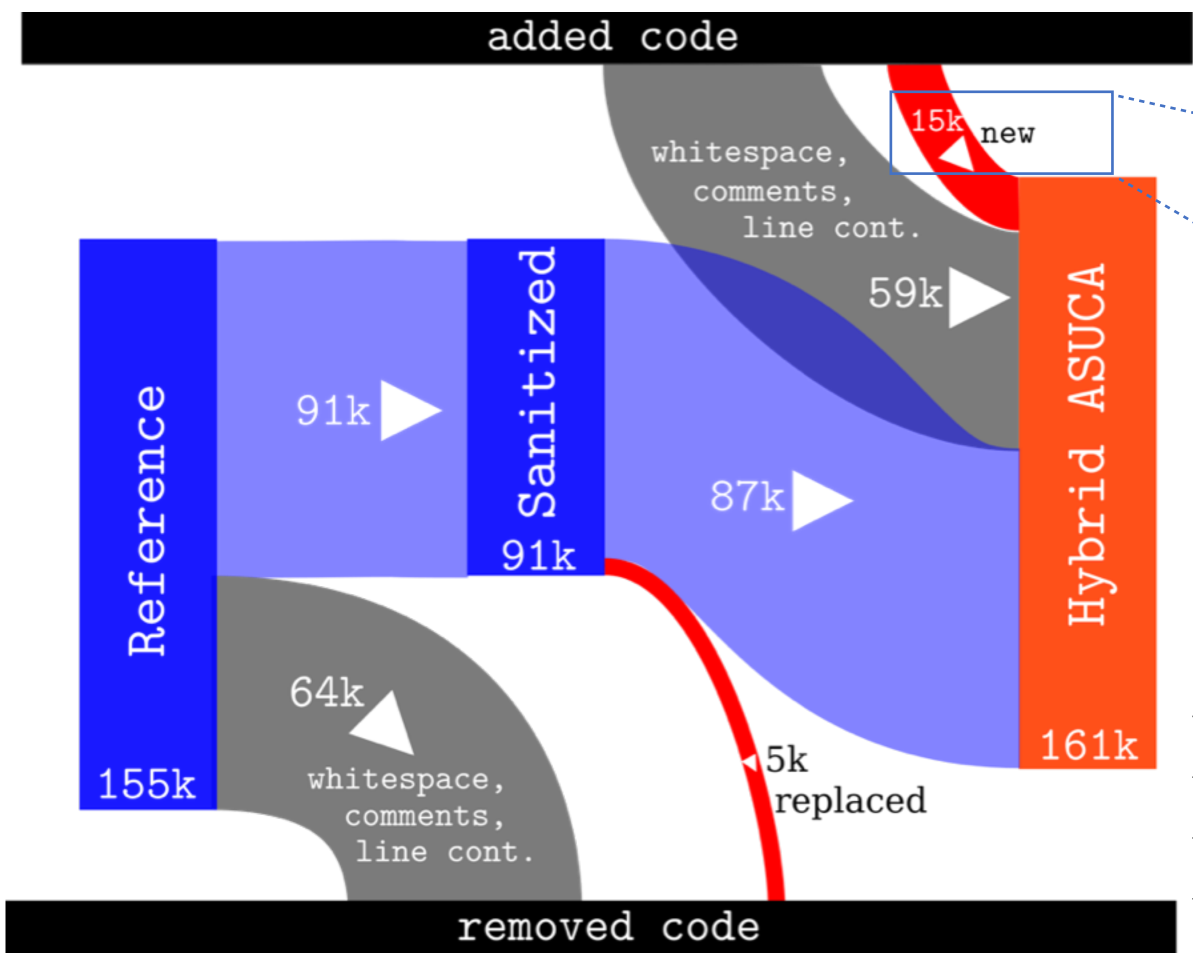


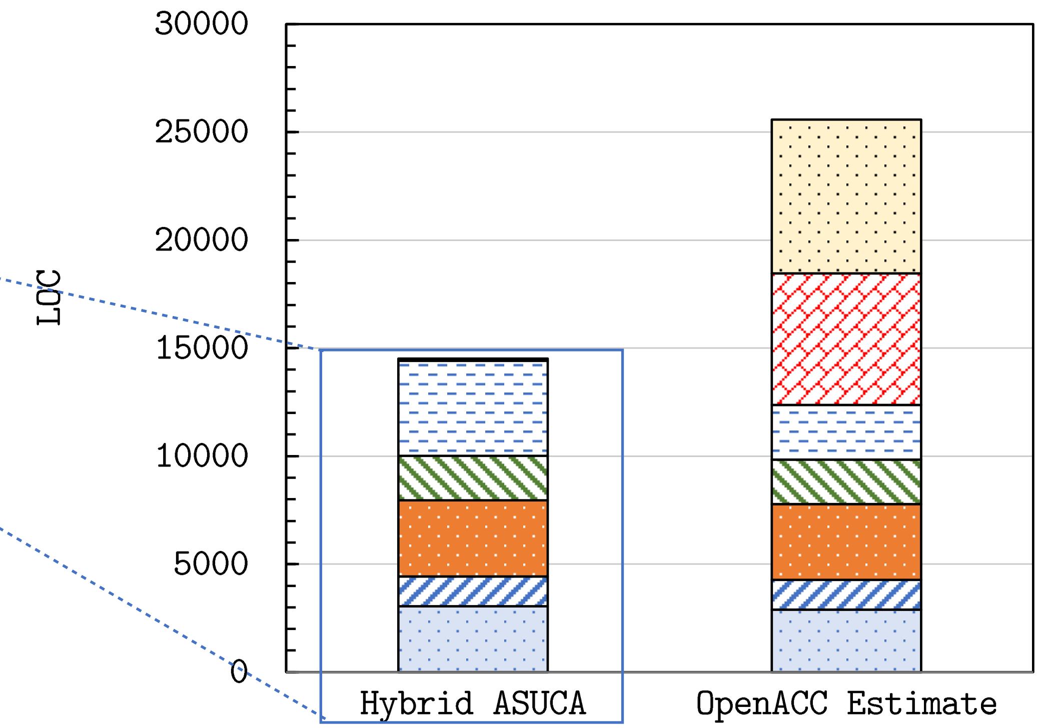
Figure 3.5: Total cloud cover result for ASUCA using 2km resolution grid and real initial data

Productivity Results

Code Reuse and Changes



Comparison with OpenACC Estimate



	Hybrid ASUCA	OpenACC Estimate
CPU-only physics	0	7122
storage order macros	116	6098
parallelization & data layout DSL	4398	2521
long-wave radiation	2059	2059
modified data spec./init	3519	3519
routine & call signatures	1381	1381
other	3046	2884

4. Performance

Performance Model: Reduced Weather Application

$$\frac{\partial T}{\partial t} = \kappa \nabla^2 T + f$$

T	temperature
κ	thermal diffusivity
f	radiation heat

- diffuse: 7-point Von-Neumann-type stencil, 0.125 FLOP/B DP
- radiate: 0.0625 FLOP/B DP

➡ memory bandwidth bounded on all architectures (e.g. system balance on P100: 7.8 FLOP/B, 6-core Westmere: 2.8 FLOP/B)

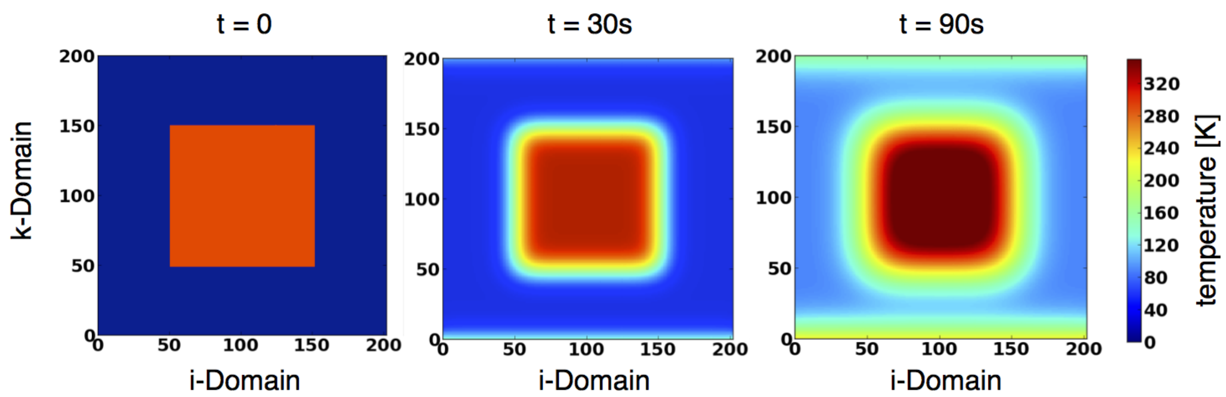
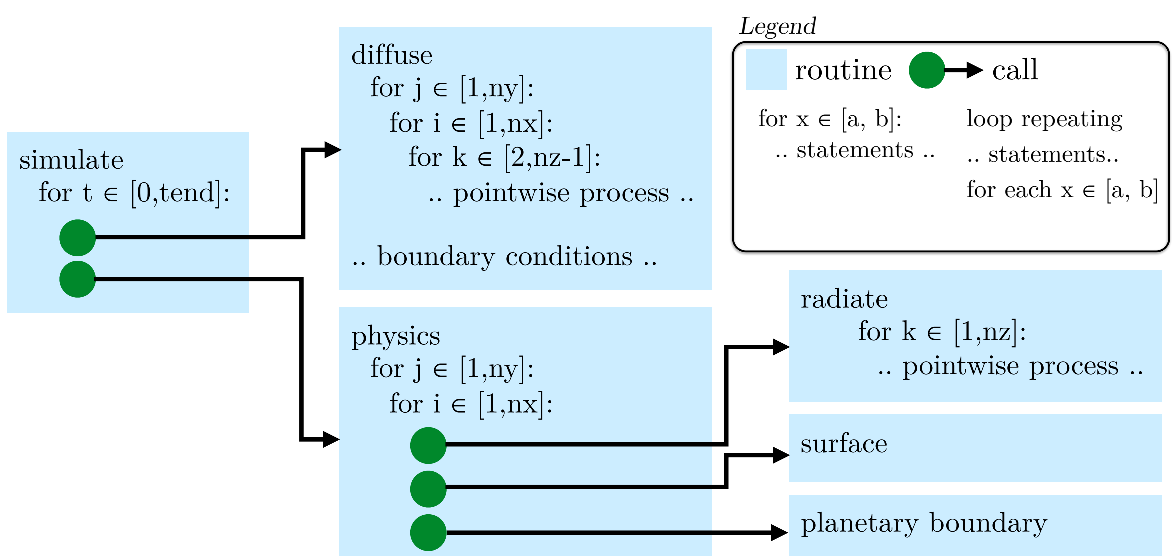


Figure 4.1: Output at $j = 100$.



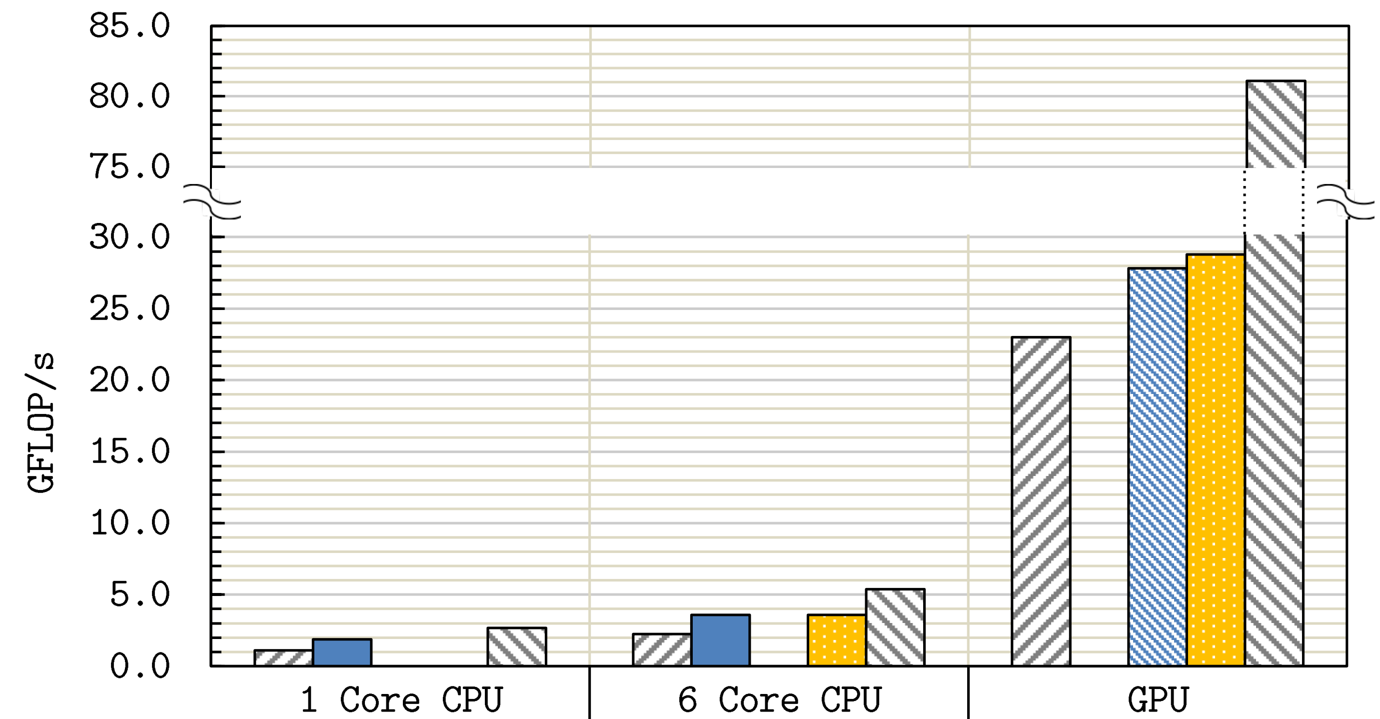
$$\boxed{t_D = \underbrace{\frac{\Delta t_{output}}{\Delta t_{timestep}}}_{\text{\#timesteps between output}} \left(n_x \cdot n_y \cdot n_z \cdot \left(\underbrace{\frac{b \cdot m_{sa}}{BW_D}}_{\text{pointwise inner diffusion}} + \overbrace{\frac{b \cdot m_{HtoD}}{BW_{HtoD}}}^{\text{pointwise host/device I/O}}\right) + n_y \cdot n_z \cdot \underbrace{\frac{m_{ra}}{RAD_D}}_{\text{pointwise diffusion boundary}}\right)}$$

Results: Reduced Weather Application

	IJK Order	KIJ Order
CPU Single Core	1.73s	1.28s
GPU (OpenACC) (Fastest Implementation)	0.10s	0.77s

Influence of storage order on execution time

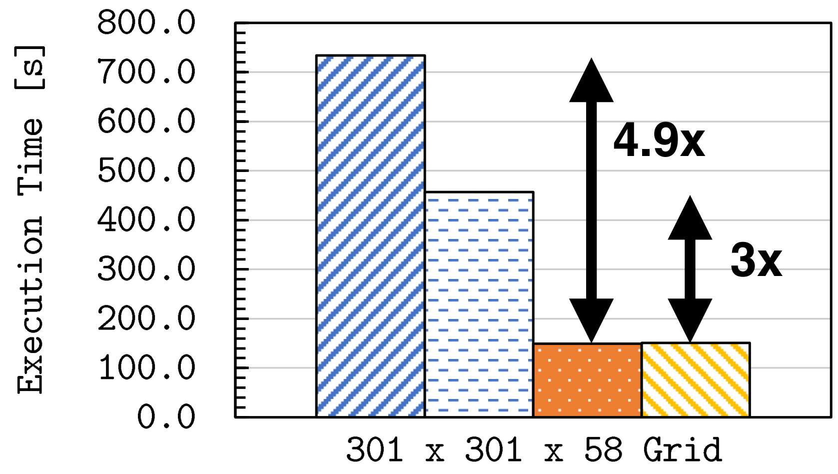
Performance of reduced weather app. for separately implemented, vs. Hybrid Fortran generated, vs. model on 256x256x256 grid, 100 timesteps (fastest implementation)



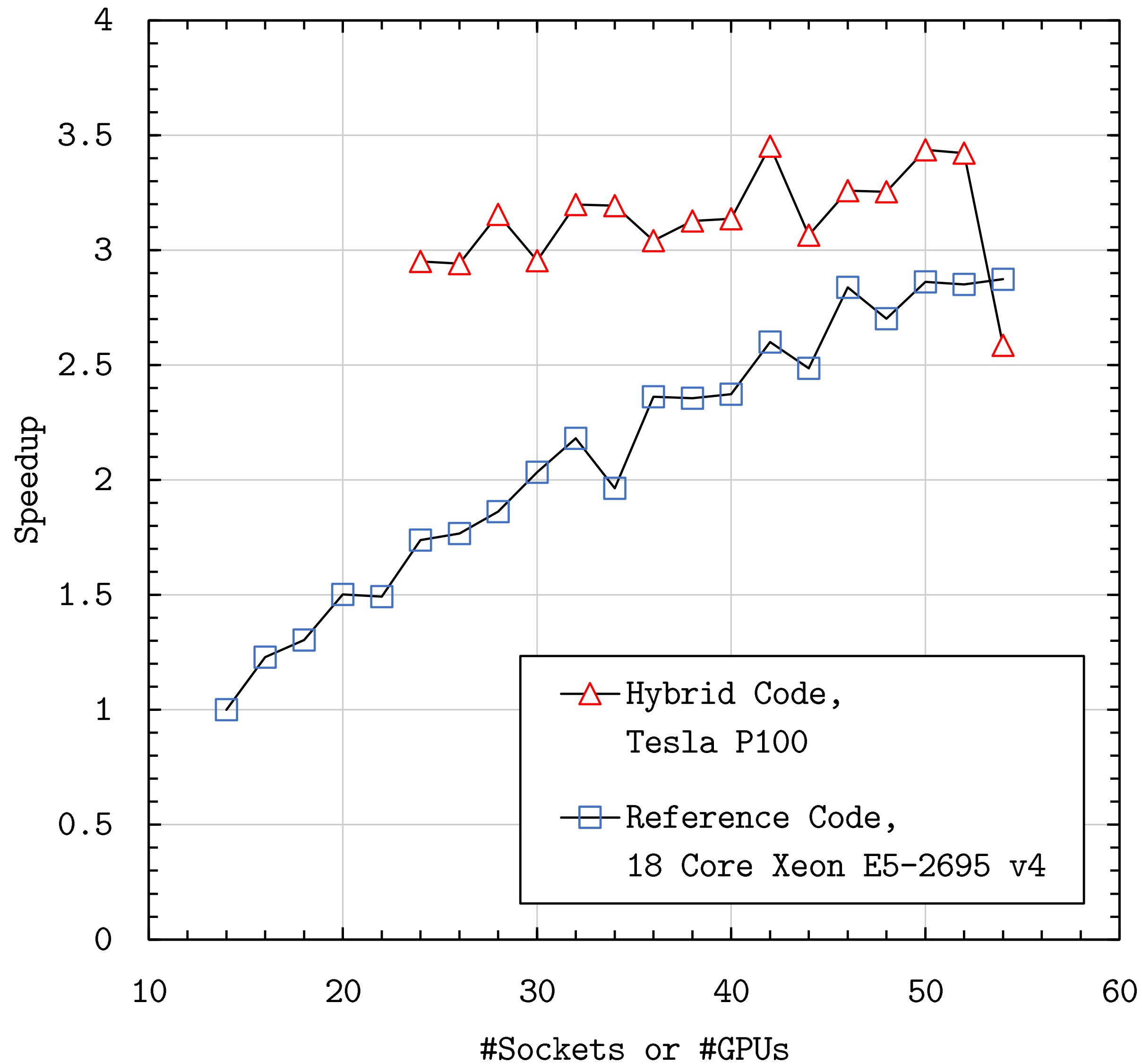
	1 Core CPU	6 Core CPU	GPU
No Cache Model	1.1	2.3	23.0
OpenMP	1.9	3.6	
OpenACC			27.8
Hybrid Fortran		3.6	28.8
Perfect Cache Model	2.7	5.4	81.1

Results: Hybrid ASUCA

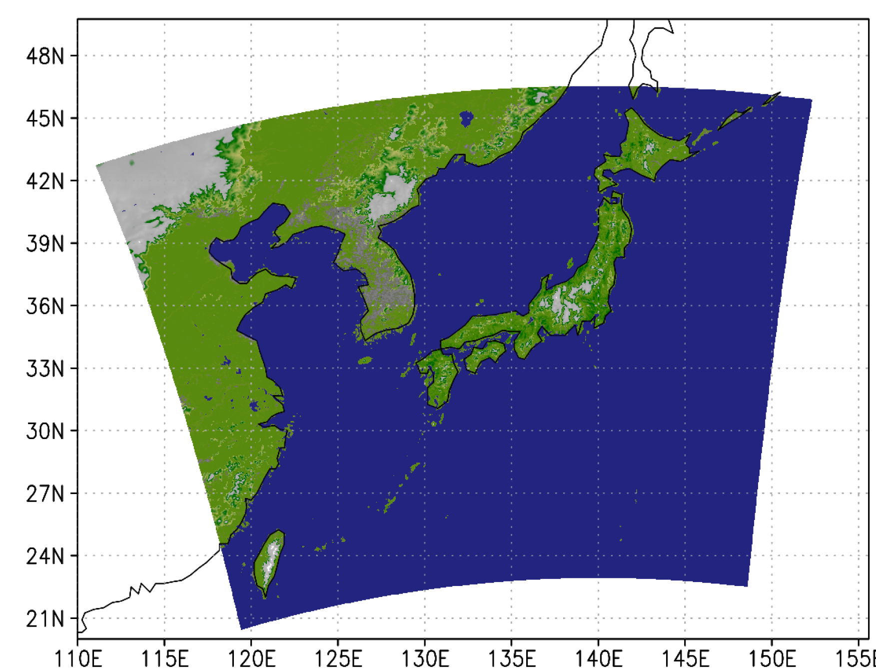
Kernel performance on reduced Grid (301 x 301 x 58) →



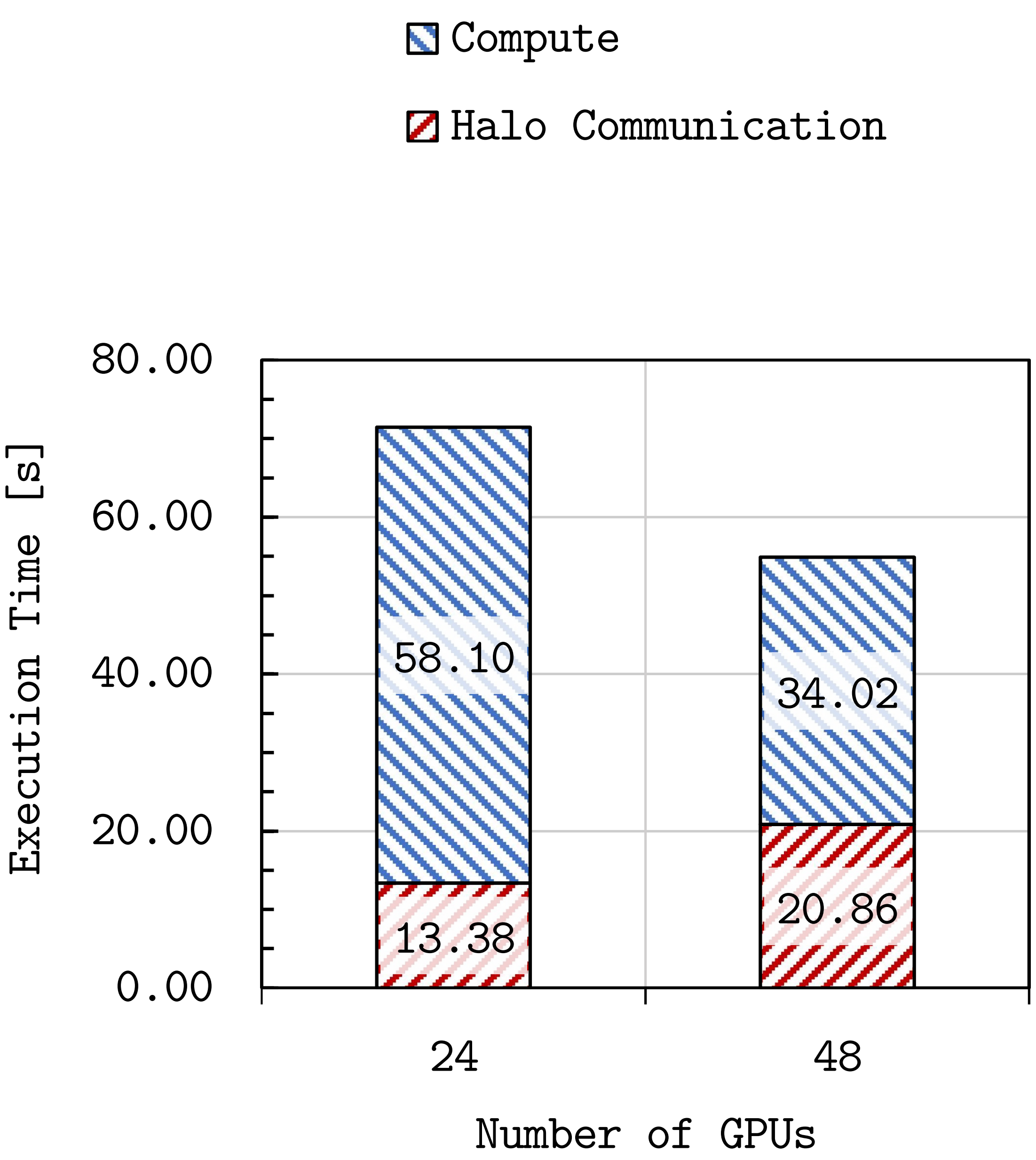
ASUCA Reference, 4 x 6-core Xeon X5670	734.0
ASUCA Reference, 1 x 18-core Xeon E5-2695 v4	456.7
Hybrid ASUCA, 4 x Tesla K20x	148.9
Hybrid ASUCA, 1 x Tesla P100	151.1

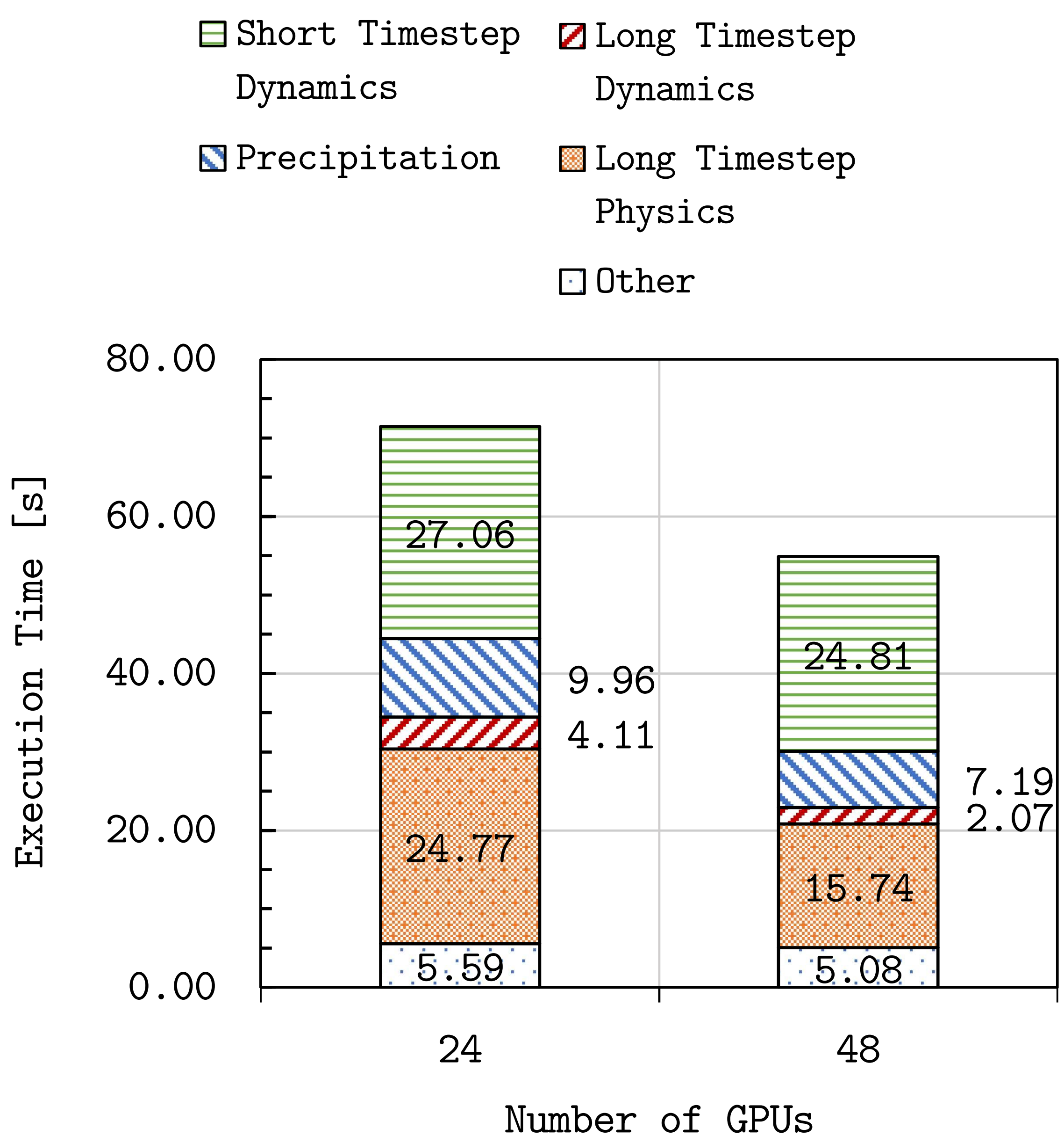


← **Strong scaling results on Reedbush-H, 1581 x 1301 x 58 Grid (Japan and surrounding region)**



Results: Hybrid ASUCA





Impact of communication and modules for strong scaling on 1581 x 1301 x 58 ASUCA Grid, using 2x P100 GPU per node (TSUBAME 3)

Contributions

- ✓ new granularity abstraction and memory layout transformation method
- ✓ applied to ASUCA, resulting in >3x speedup in kernel performance and >2x reduction in processors required for a full scale run with real data
- ☐ **method unique in increasing productivity for porting coarse-grained codes to GPU**

5. Method Comparison

- Outline of Methods
- Productivity Characterization

Methods for Hybrid CPU/GPU HPC Codes

Domain-Specific Languages

Parallelization, data access patterns and potentially other program aspects are abstracted, requiring a full rewrite:

- Shimokawabe et al., STELLA, GridTools
  - C++ user language
  - Memory access patterns abstracted (stencil DSL)
  - Include communicator
- Atlas
  - C++ and Fortran user languages
  - Higher level abstraction, code applies to variable grids

Directive-Based Methods

Parallelization and data movement are abstracted, access patterns are fixed:

- OpenACC used directly in various degrees by
  - Lapillonne et al.
  - Govett et al.
  - Norman et al.

Granularity Optimization

The following approaches to code granularity optimization are known:

- Kernel fusion is employed in the following approaches:
  - STELLA / GridTools
  - CLAW compiler
- Proposed Hybrid Fortran is a unique new method to abstract granularity

Memory Layout Transformation

Allows variable memory layouts without a full code rewrite:

- Kokkos
  - C++ user language
- ICON
  - Fortran user language
- Hybrid Fortran

Method Characterization

method	memory layout	grid
Shimokawabe et al.	abstracted	fixed
STELLA & GridTools	abstracted	fixed
Atlas	abstracted	variable
OpenACC & OpenMP	fixed	fixed
CLAW plus OpenACC	fixed	fixed
Kokkos	transforming	fixed
ICON	transforming	fixed
Hybrid Fortran	transforming	fixed

Data structure characterization

Control structure characterization

method	parallelization	granularity	communication	language
Shimokawabe et al.	abstracted	fixed	abstracted	C++
STELLA & GridTools	abstracted	transforming (kernel fusion)	abstracted	C++
Atlas	abstracted	fixed	abstracted	C++ and Fortran
OpenACC & OpenMP	transforming	fixed	fixed	C++ and Fortran
CLAW plus OpenACC	transforming	transforming (kernel fusion)	fixed	Fortran
Kokkos	abstracted	fixed	fixed	C++
ICON plus OpenMP	transforming	fixed	fixed	Fortran
Hybrid Fortran	abstracted	abstracted	fixed	Fortran

Method Evaluation

$$P_\alpha(r_\alpha) = 1 - \overbrace{A_{ij}(\alpha)}^{\text{investment scoring matrix}} \cdot (\overbrace{w_\alpha}^{\text{weights}} \cdot diag(\overbrace{r_\alpha}^{\text{requirement vector}}))^T$$

$$P(r) = mean(P_{data}(r_{data}), P_{control}(r_{control}))$$

method	**Xmas tree**	**phys.**	**dyn. (Fortran)**	**dyn. (C++)**
Shimokawabe et al.	0.35	0.46	0.54	0.79
STELLA & GridTools	0.47	0.55	0.54	0.79
Atlas	0.67	0.69	0.79	0.79
OpenACC & OpenMP	0.31	0.36	0.41	0.41
CLAW plus OpenACC	0.31	0.45	0.41	0.15
Kokkos	0.39	0.63	0.64	0.90
ICON	0.39	0.86	0.91	0.65
Hybrid Fortran	0.52	0.98	0.90	0.64

Table 5.5: Productivity score $P(r)$ for different usecases r.

Interactive Evaluation Matrix

Matrix is [publicly accessible](#).

1

	Your legacy application contains...	
1	... C/C++ code.	Yes
2	... Fortran code.	No
3	... a fine-grained kernel code structure that would benefit from coarser granularity on some architectures.	No
4	... a coarse-grained kernel code structure (presumably requiring refinement for GPU).	No
5	... a rewrite of inter-processor communication (presumably for a new network architecture)	Yes
6	... a memory layout that requires a different ordering on GPU (to achieve a satisfying performance, i.e. coalesced memory access).	Yes
7	... a grid that requires a geometry change, or multiple Grids.	No

2

Data Structure Expressiveness →

Language & Control Structure Expressiveness ↓

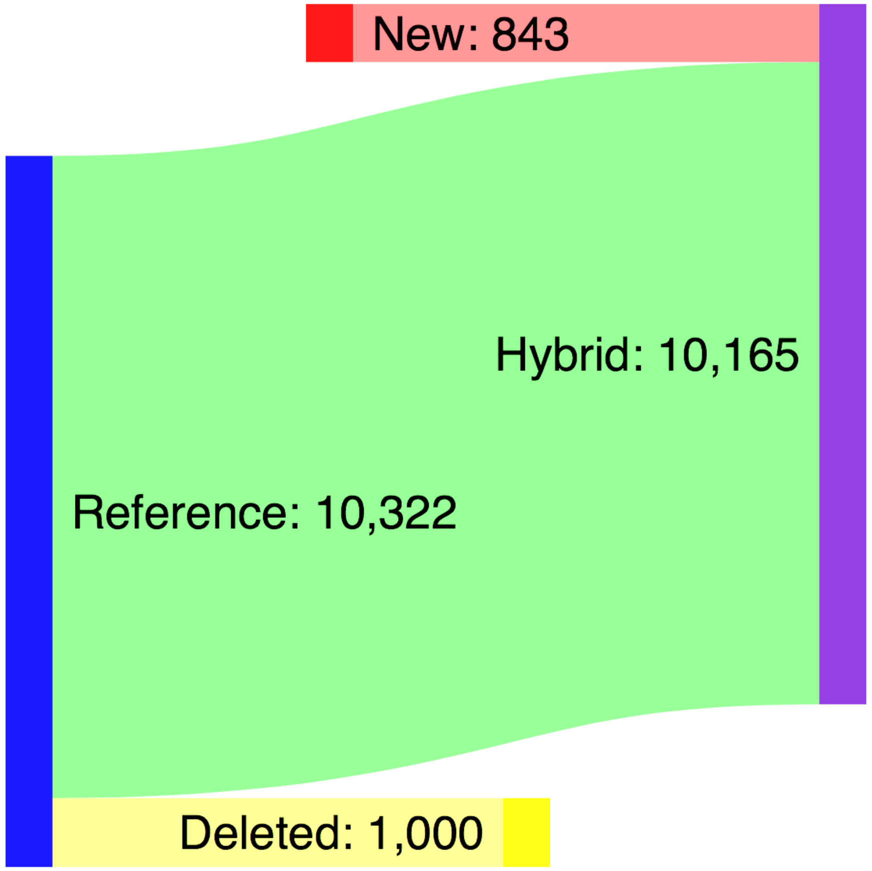
Productivity landscape mapped to porting methods
Productivity values calculated as the product of Language & control structure productivity score and data structure productivity score

Memory Layout				fixed	abstracted	abstracted	transforming	transforming	
Grid				fixed	fixed	variable	fixed	variable	
User Language	**Code Granularity**		**Parallelization**						**Prod. Score ↓**
C++	fixed		fixed	0.01	0.03	0.09	0.04	0.14	14%
			abstracted	0.02	0.06	0.21	0.11 RAJA Kokkos	0.33	33%
			transformi	0.02	0.06	0.22	0.11	0.34	34%
	abstracted		abstracted	0.04	0.15	0.52	0.26	0.81	81%
		fine to coarse		0.0	0.13 STELLA GridTools Legion	0.47	0.23	0.73	73%
		coarse to	abstracted	0.03	0.14	0.48	0.24	0.76	76%
				0.04	0.15	0.52	0.26	0.81	81%
				0.03	0.14	0.48	0.24	0.76	76%
				0.04	0.14	0.50	0.25	0.79	79%
				0.04	0.16	0.55	0.27	0.86	86%
				0.01	0.03	0.09	0.04	0.14	14%
				0.02	0.06	0.21	0.11	0.33	33%
				0.02 F2CACC	0.06	0.22	0.11 ICON DSL+OpenMP	0.34	34%
				0.04	0.15	0.52	0.26 Hybrid Fortran	0.81	81%
				0.03	0.13	0.47	0.23	0.73	73%
				0.03	0.14	0.48	0.24	0.76	76%
				0.04	0.15	0.52	0.26	0.81	81%
				0.03 CLAW+OpenACC	0.14	0.48	0.24	0.76	76%
		coarse to fine	transforming	0.04	0.14	0.50	0.25	0.79	79%
		two-way		0.04	0.16	0.55	0.27	0.86	86%
C++ & Fortran	fixed		fixed	0.01 CUDA	0.05	0.18	0.09	0.28	28%
			abstracted	0.02	0.09	0.30 Atlas	0.15	0.47	47%
			transforming	0.02 OpenACC OpenMP	0.09	0.31	0.15	0.48	48%
	abstracted		abstracted	0.04	0.17	0.61	0.30	0.95	95%
		fine to coarse		0.04	0.16	0.55 GGDML	0.28	0.87	87%
		coarse to fine		0.04	0.16	0.57	0.29	0.90	90%
		two-way	abstracted	0.04	0.17	0.61	0.30	0.95	95%
	transforming	fine to coarse		0.04	0.16	0.57	0.29	0.90	90%
		coarse to fine		0.04	0.17	0.59	0.30	0.93	93%
		two-way	transforming	0.05	0.18	0.64	0.32	1.00	100%
			Prod. Score →	**5%**	**18%**	**64%**	**32%**	**100%**	

Example Application: NICAM Physics

- Cloud microphysics
- Precipitation of rain, snow, graupel
- 111 loops to parallelize
- Due to timing issues and influenza: Roughly one week to work on this benchmark
- Hours logged: ~31.3.

number of lines of code:



	Runtime [s]
Reference, 2x 14-core Broadwell [1]	**0.595**
Hybrid, 2x 14-core Broadwell [2]	**0.941**
Hybrid, 1x P100 GPU [3]	**0.232**

6. Conclusion

Summary

Background

- ✓ paradigm shift towards throughput oriented design
- ✓ GPUs attractive for NWP (high mem. bandwidth)
- ✓ productivity and maintainability of GPU approaches lacking

Motivation

- ✓ Many of today's **NWP- and climate models cannot make efficient use of high-throughput architectures**. **We want to find** and prove **easily adoptable approach**.

Goal

- ✓ **GPU** port for "**ASUCA**" **NWP model** in Fortran with **minimal code divergence** / **minimal learning**

Contributions

- ✓ **new granularity abstraction** and **memory layout transformation method**
- ✓ applied to ASUCA, resulting in **>3x speedup in kernel performance** and **>2x reduction in processors** required for a full scale run with real data
- ✓ **method unique** in **increasing productivity** for porting **coarse-grained codes** to GPU

On all previous projects applying high-throughput architectures to NWP and climate models [27]:

"All these approaches were effectively addressing fine-grained parallelism in some way or other without addressing coarser grained concurrency, and all involved various levels of "intrusion" into code, from adding/changing codes, to complete rewrites or translations."

Prof. Bryan Lawrence
Professor of Weather and Climate Computing
Director of Models and Data @ NCAS

[27] Lawrence, Bryan N., et al. "Crossing the Chasm: How to develop weather and climate models for next generation computers?", under review for Geosci. Model Dev. (2017).

On how ACME model (DOE) cannot share a single source code for CPU and GPU due to register pressure[16]:

"The only remedy for this at present is to break the kernel up into multiple kernels. (…) On the CPU one would want to keep an element loop fused together for caching reasons."

Dr. Matthew R. Norman
Computational Climate Scientist
Oak Ridge National Laboratory

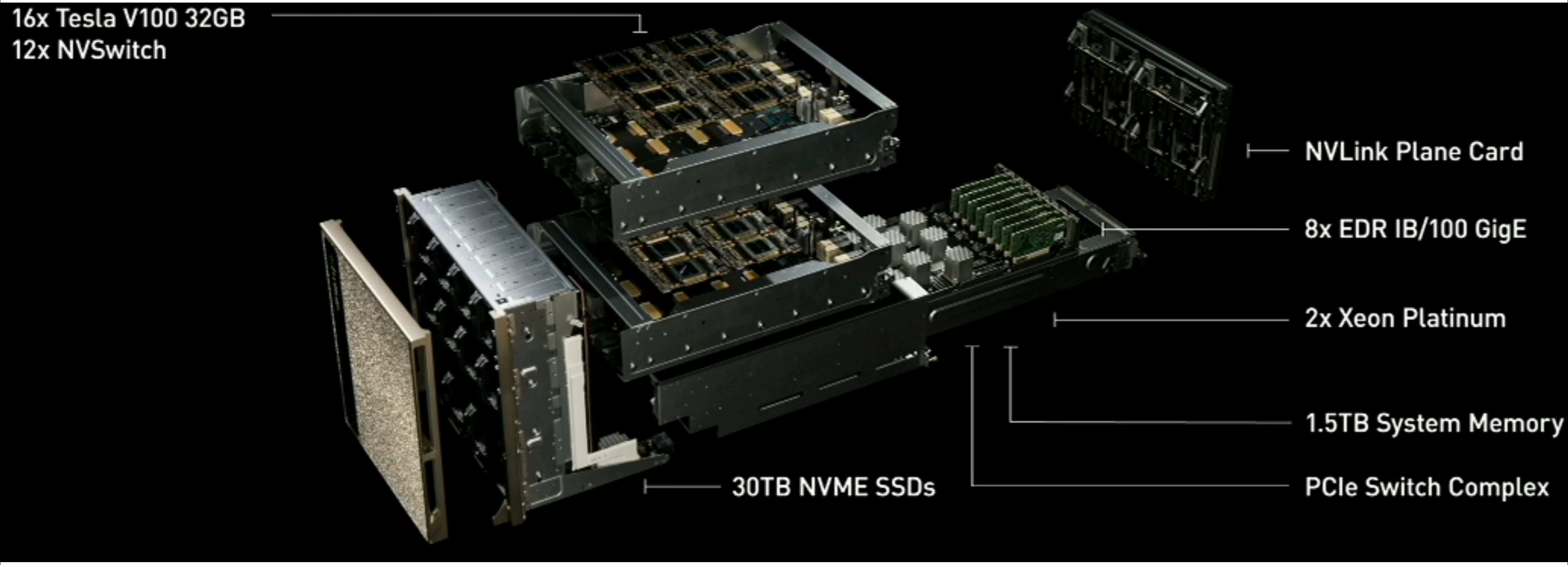
[16] Norman, Matthew R., Azamat Mametjanov, and Mark Taylor. "Exascale Programming Approaches for the Accelerated Model for Climate and Energy." (2017).

Concluding Remarks

- All **previous projects** porting NWP and climate models to high-throughput architectures had to **choose between**
    - complete rewrite (**maximum learning**),
    - code divergence (**poor maintainability**),
    - efficiency loss on at least one architecture (**poor performance**).
- **This work** shows a **new approach**, which has **many potential applications** beyond GPU and beyond NWP.
    - Hybrid Fortran is Open Source and can be applied directly where suitable.
    - Method as documented can be replicated in other applications, even if Hybrid Fortran is not used.

Outlook

- NVIDIA introduced DGX-2 - a **400k USD GPU system**
- Thesis: **Operational 2km ASUCA** on a **single DGX-2** possible
  - 16x Tesla V100s totaling 512GB HBM with unified address space
  - Halo communication entirely through 900 GB/s NVSwitch



Thank you for your attention.

[1] Bjerknes, Vilhelm. "Das Problem der Wettervorhersage, betrachtet vom Standpunkte der Mechanik und der Physik." *Meteor. Z.* 21 (1904): 1-7.
[2] Woolard, Edgar W. "LF Richardson on weather prediction by numerical process." *Monthly Weather Review* 50.2 (1922): 72-74.
[3] Lynch, Peter. “Richardson’s forecast: What went wrong?” *NOAA NWP* 50 (2004).
[4] Courant, Richard, Kurt Friedrichs, and Hans Lewy. "Über die partiellen Differenzengleichungen der mathematischen Physik." *Mathematische annalen* 100.1 (1928): 32-74.
[5] Charney, Jules G., Ragnar Fjörtoft, and J. von Neumann. "Numerical integration of the barotropic vorticity equation." *Tellus* 2.4 (1950): 237-254.
[6] White, Andy A., et al. "Consistent approximate models of the global atmosphere: shallow, deep, hydrostatic, quasi-hydrostatic and non-hydrostatic." *Quarterly Journal of the Royal Meteorological Society* 131.609 (2005): 2081-2107.
[7] Kurowski, Marcin J., Wojciech W. Grabowski, and Piotr K. Smolarkiewicz. "Anelastic and compressible simulation of moist deep convection." *Journal of the Atmospheric Sciences* 71.10 (2014): 3767-3787.
[8] Ishida, Junichi, et al. "Development of a new nonhydrostatic model ASUCA at JMA." *CAS/JSC WGNE Research Activities in Atmospheric and Oceanic Modelling* 40 (2010): 0511-0512.
[9] Dennard, Robert H., et al. "Design of ion-implanted MOSFET's with very small physical dimensions." *IEEE Journal of Solid-State Circuits* 9.5 (1974): 256-268.
[10] Kuhn, Kelin J. "Moore's Law Past 32nm: Future Challenges in Device Scaling." *Computational Electronics, 2009. IWCE'09. 13th International Workshop on*. IEEE, 2009.
[11] Garland, Michael, and David B. Kirk. "Understanding throughput-oriented architectures." *Communications of the ACM* 53.11 (2010): 58-66.
[12] Michalakes, John, and Manish Vachharajani. "GPU acceleration of numerical weather prediction." *Parallel Processing Letters* 18.04 (2008): 531-548.
[13] Govett, Mark, Jacques Middlecoff, and Tom Henderson. "Directive-based parallelization of the NIM weather model for GPUs." *Accelerator Programming using Directives (WACCPD), 2014 First Workshop on*. IEEE, 2014.
[14] Shimokawabe, Takashi, Takayuki Aoki, and Naoyuki Onodera. "High-productivity framework on GPU-rich supercomputers for operational weather prediction code ASUCA." *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014.
[15] Fuhrer, Oliver, et al. "Towards a performance portable, architecture agnostic implementation strategy for weather and climate models." *Supercomputing frontiers and innovations* 1.1 (2014): 45-62.
[16] Norman, Matthew R., Azamat Mametjanov, and Mark Taylor. "Exascale Programming Approaches for the Accelerated Model for Climate and Energy." (2017).

References

[17] Briegleb, Bruce P. "Delta-Eddington approximation for solar radiation in the NCAR Community Climate Model." *Journal of Geophysical Research: Atmospheres* 97.D7 (1992): 7603-7612.
[18] Goody, R. M. "A statistical model for water-vapour absorption." *Quarterly Journal of the Royal Meteorological Society* 78.336 (1952): 165-169.
[19] Kiehl, J. T., and Charles S. Zender. "A prognostic ice water scheme for anvil clouds." *WMO Publications TD* (1995): 167-188.
[20] Kaufman, Y. J., et al. "Absorption of sunlight by dust as inferred from satellite and ground-based remote sensing." *Geophysical Research Letters* 28.8 (2001): 1479-1482.
[21] Nakanishi, Mikio, and Hiroshi Niino. "An improved Mellor–Yamada level-3 model with condensation physics: Its design and verification." *Boundary-layer meteorology* 112.1 (2004): 1-31.
[22] Beljaars, A. C. M., and A. A. M. Holtslag. "Flux parameterization over land surfaces for atmospheric models." *Journal of Applied Meteorology* 30.3 (1991): 327-341.
[23] Clément, Valentin. "CLAW Fortran Compiler Documentation". 2017
[24] Lapillonne, Xavier, and Oliver Fuhrer. "Using compiler directives to port large scientific applications to GPUs: An example from atmospheric science." *Parallel Processing Letters* 24.01 (2014): 1450003.
[25] Edwards, H. Carter, Christian R. Trott, and Daniel Sunderland. "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns." *Journal of Parallel and Distributed Computing* 74.12 (2014): 3202-3216.
[26] Torres, Raul, et al. "ICON DSL: A domain-specific language for climate modeling." *International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, Colo.* 2014.
[27] Lawrence, Bryan N., et al. "Crossing the Chasm: How to develop weather and climate models for next generation computers?", under review for Geosci. Model Dev. (2017).

References

NWP Models

Approximations:

- **Spherical-geopotential** (G): Gravity without horizontal component
- **Shallow-atmosphere** (S)
  - ➡ Gravitation constant with distance from surface
  - ➡ Finer vertical vs. horizontal resolution (aspect ratio)
  - ➡ Mixed implicit/explicit iteration schemes used to avoid inefficiently short time steps

- **Hydrostatic** (H): Atmosphere horizontally compressible, vertically incompressible
  - ➡ Sound waves filtered

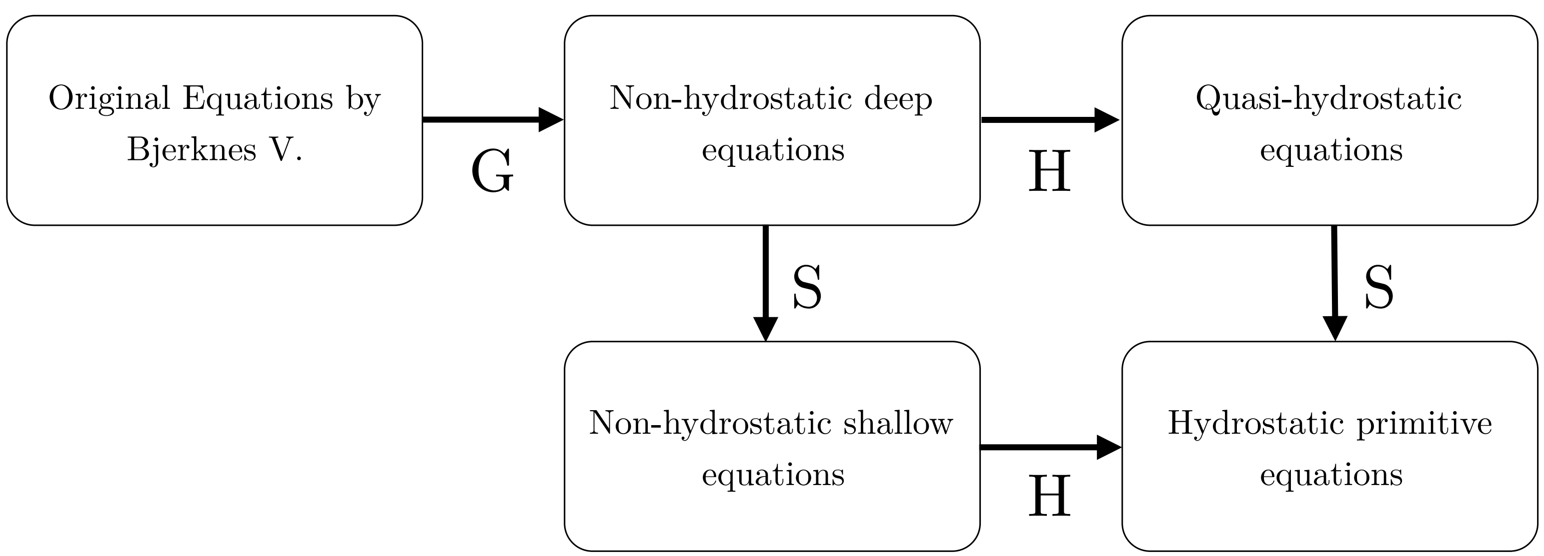


Figure 1.1: Interrelations of atmospheric models with respect to their approximations according to White et al.

ASUCA NWP Model

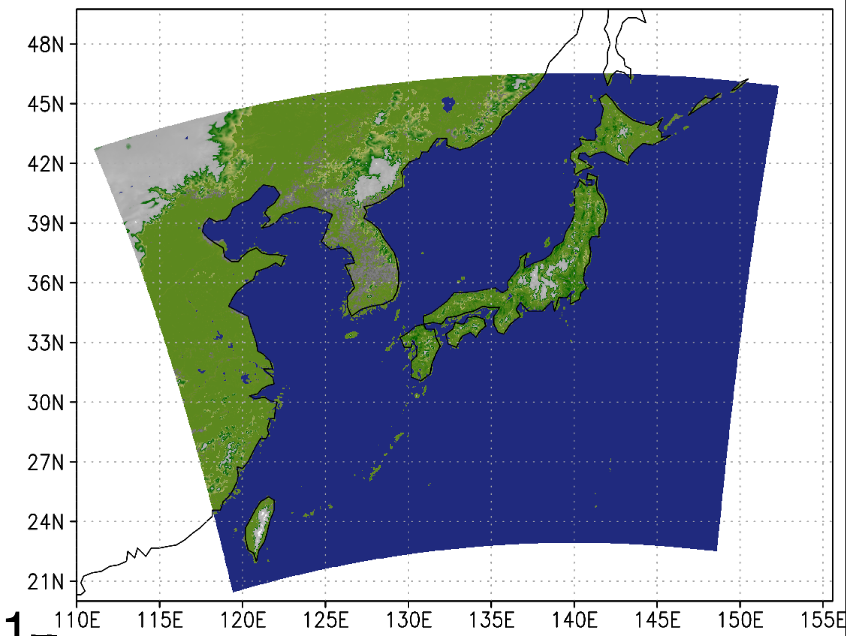


Figure 1.2: ASUCA’s model simulation boundaries

What is ASUCA**?**

- ``**A**suca is a **S**ystem based on a **U**nified **C**oncept for **A**tmosphere''
- **fully compressible**, **non-hydrostatic** weather prediction model
- **regional scale** - as depicted in Figure 1.2
- one of **main operational** forecast models in Japan, **in production** since 2014
- spatial discretization: **finite-volume method** on **Arakawa-C-type rectangular** grid
  - k-coordinates are **terrain-following**
  - **general** horizontal **coordinates**, with lat/lon and Lambert conformal conic projections available
- time discretization:
  - **third-order Runge-Kutta** based iteration scheme for **advection and Coriolis** force
  - **time-splitting method**, employing secondary third-order Runge-Kutta iteration with **short time step** for **sound-** and **gravity waves**
- **vertical advection** of water substances solved using **separate time step** for **each column** using **separate** Courant-Friedrichs-Lewy **convergence** condition
- **vertical-only** models for parametrization of **radiation**, **planetary boundary layer** and **surface physical processes**

NWP and Computational Performance

- As approximations show, available comp. performance has strong impact on design of NWP models.
- In Earth system models differentiate between **dynamical**- and **physical** processes.
  - *dynamics*: phenomena large enough to model in-grid.
  - *physics:* phenomena too small for spatial grid resolution. Separate models are computed, generating tendency values for dynamical time iteration (parametrization).
- Increase in comp. performance allows increasing grid resolution.
  - ➡ Physical processes slowly migrate towards dynamical modelling.
  - During last decades this mainly applies to resolution of increasingly small cloud formations in dynamical core.
- Typically applied finite-volume and finite-difference based discretization methods are bottlenecked by memory bandwidth in the dynamics.

➡ Progress in comp. performance and thus grid resolution leads to increasing memory bandwidth pressure.

GPUs for Numerical Weather Prediction

- **GPU**s offer **high memory bandwidth**, which is in high demand in NWP.
  ➡ GPUs are an **attractive target** architecture.
- Major **problems** to solve for existing regular grid NWP codes:
  - **Memory layout** needs change
  - **Code granularity** in physical processes too coarse for GPU
  - Extending **device data region** to entire time integration
    - Requires GPU port of all processes run in simulation
    - Ensures minimal communication across slow bus between host and device
- **Existing methods** to solve these problems:
  - Only apply GPU to dynamical core or smaller parts of physics.
  - Rewrite Fortran code using C++ templates for architecture specialization.
  - Code divergence between CPU and GPU to solve granularity issues.

➡ **Unsatisfactory** to maintain a unified, coherent and efficient code base in Fortran (the standard in NWP)

➡ **For ASUCA, a solution with none of these drawbacks was sought.**

GPU Computing - Programming Model

- **CPU vector programming:** single instruction, multiple data (**SIMD**)
  - Vectorization highly **sensitive** to **data dependent branching** and **loop ordering** - example shown below difficult or impossible to vectorize
- **GPU**s: single instruction, multiple threads (**SIMT**)
  - **Branching**, **early returns** and backwards jumps (**inner loops**) **supported** for each thread in hardware architecture
  - Vectorization thus **insensitive to loop ordering and branching**

- **GPU**s **do not support** real **context switching within kernels** - all function calls are inlined, thus share register scope.
  - ➡ Due to register pressure as well as practicality, **fine-grained kernels** are **required** on GPU.

```fortran
do j = 1, ny
  do i = 1, nx
    if (b(i,j)) then
        do k = nz - 1, 1, -1
            a(k,i,j) = a(k+1,i,j) * exp(- sqrt(gamma(k,i,j)) * tau)
        end do
    else
        do k = 1, nz - 1
            a(k,i,j) = 0.0d0
        end do
    end if
  end do
end do
```

GPU Computing - Peak Performance

- **Recent CPU**s have **caught up** in **theoretical throughput** - theoretical GFLOP/s per Watt of some CPUs now two thirds of current GPUs.
- However: **Memory bandwidth** shows a **clear advantage for GPU** - e.g. an 8.2x advantage in peak bandwidth per Dollar for latest HPC targeted models.
  (advantage even stronger for GPU if we include memory pricing in calculation)

Characteristic	CPU	GPU
Vector length (double precision)	16	32
Core- or SM count	28	60
Clock frequency	2.5 GHz	1.3 GHz
Memory bandwidth	119 GB/s	720 GB/s
Thermal design power	205W	300W
Die size	$\sim 700\ mm^2$	$610\ mm^2$
List price	\$10,009	\$7374 (including 16 GB HBM2 Memory)

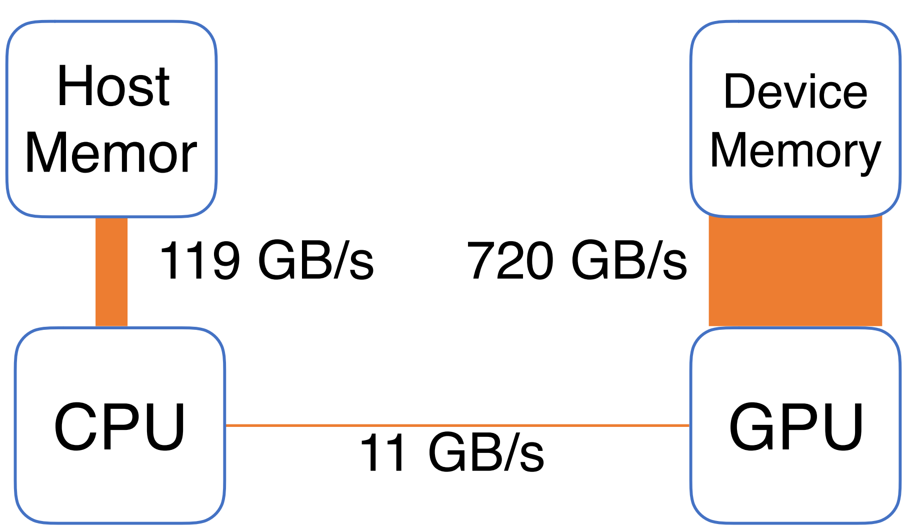
Metric	CPU	GPU
Peak GFLOP/s (double precision)	2240	5004
GFLOP/s per Watt	10.9	16.7
GFLOP/s per Dollar	0.22	0.68
Memory bandwidth per Dollar	11.89 MB/s	97.64 MB/s
FLOP/Byte system balance	18	6

Tables 1.1 and 1.2: Intel Xeon 8180 (Skylake-SP) vs. NVIDIA Tesla P100 (Pascal)

$$P_{peak} = \frac{N_{cores} \cdot l_{vector} \cdot f_{clock}}{CPI_{min}}$$

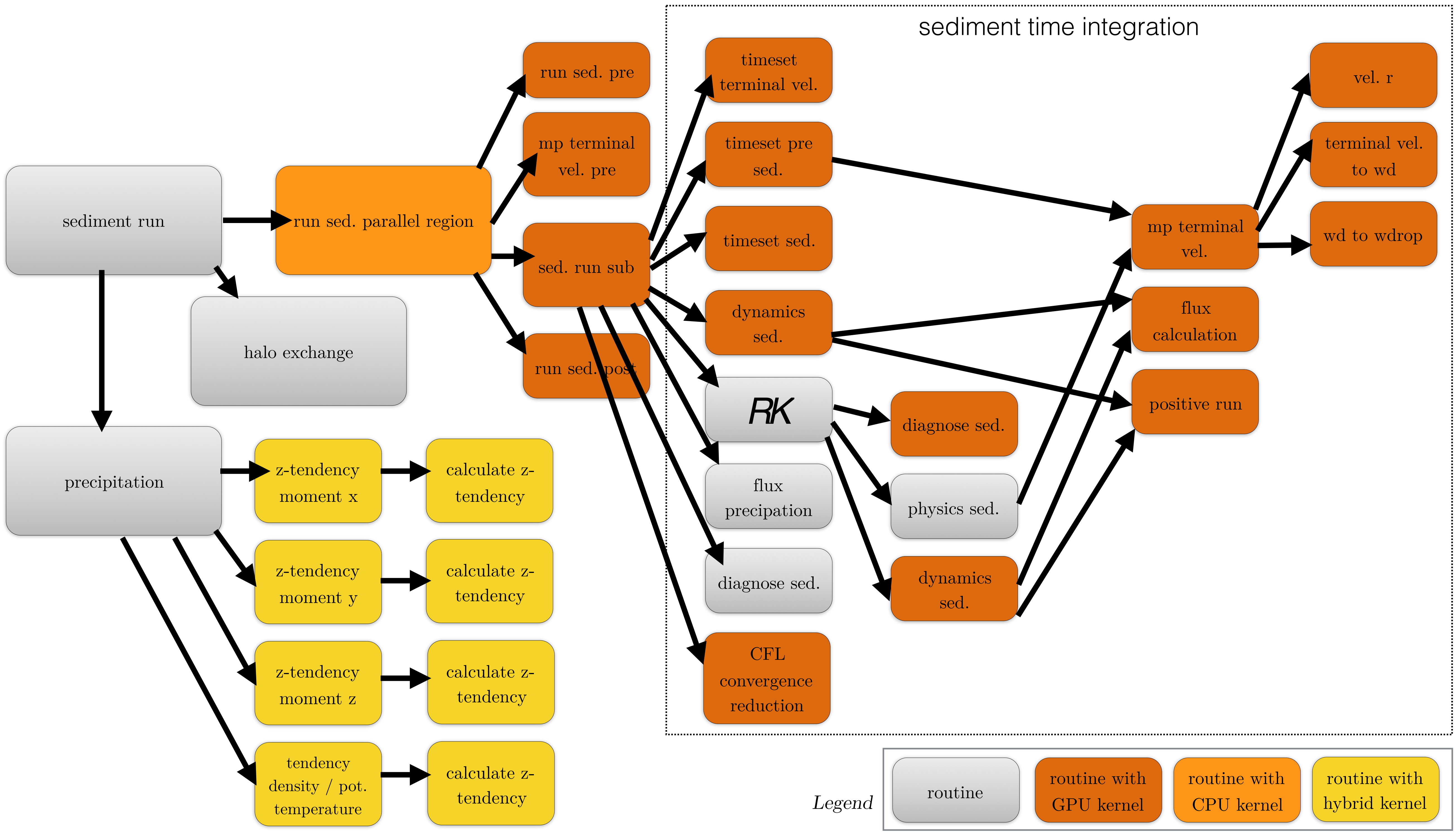
GPU Computing - Device Memory

- GPU comes with separate memory system to enable high bandwidth.
  - For applications to achieve a high performance, programmer generally needs to keep track about memory a variable resides - e.g. OpenACC data directives or CUDAMemCopy instructions.
- GPUs are particular about what order memory should be accessed in order to allow coalescence. Innermost ***parallel*** thread index (i in below example) should be mapped to unit stride.
  - This stands in contrast to CPUs where innermost loop index (k in below example) may be optimal for unit stride.
  - Performant storage order may differ between CPU and GPU.




```
do j = 1, ny
  do i = 1, nx
    if (b(i,j)) then
        do k = nz - 1, 1, -1
            a(k,i,j) = a(k+1,i,j) * exp(- sqrt(gamma(k,i,j)) * tau)
        end do
    else
        do k = 1, nz - 1
            a(k,i,j) = 0.0d0
        end do
    end if
  end do
end do
```

Column-wise Courant-Friedrichs-Lewy Convergence

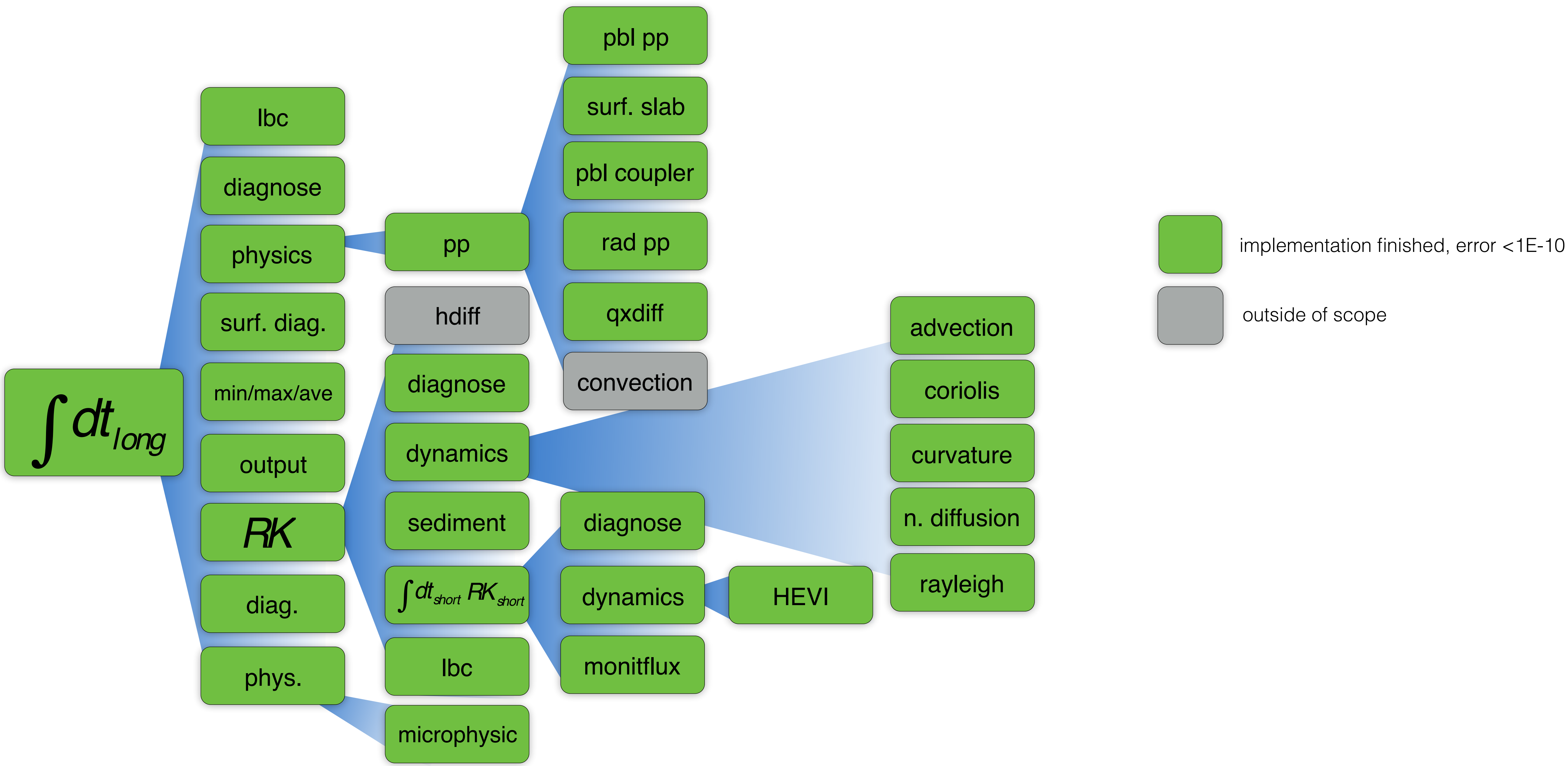


Physical Processes

- Original physical process library from JMA adapted for GPU (MSM0705 model):
  - Radiation based on 18-band model by Briegleb [17].
    - Optical cloud absorption based on statistical model by Goody [18] with thin cloud correction by Kiehl and Zehnder [19].
    - Transmission function for particle absorption uses look-up table method from empirical data gathered by NASA Goddard [20].
    - For efficient use of GPU, memory footprint of indirect radiation effects was reduced by 10x by using ad-hoc computations for each long-wave band rather than storing temporary data of all bands.
  - A Mellor-Yamada based planetary boundary layer model, improved by Nakanishi and Hiino, is adopted [21].
  - Wind momentum-, sensible heat- and latent heat surface fluxes are simulated based on Beljaars and Holtstag model [22].
- Kessler-type warm rain model is implemented for GPU.
- Hybrid Fortran's adaptive parallelization granularity used to generate GPU version.

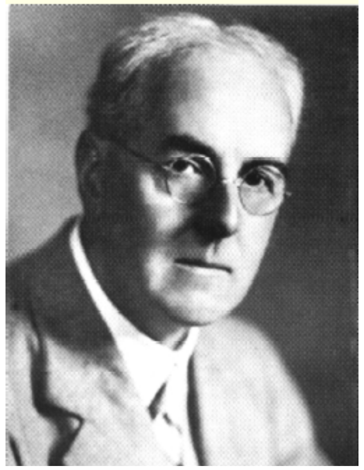
[17] Briegleb, Bruce P. "Delta-Eddington approximation for solar radiation in the NCAR Community Climate Model." *Journal of Geophysical Research: Atmospheres* 97.D7 (1992): 7603-7612.
[18] Goody, R. M. "A statistical model for water-vapour absorption." *Quarterly Journal of the Royal Meteorological Society* 78.336 (1952): 165-169.
[19] Kiehl, J. T., and Charles S. Zender. "A prognostic ice water scheme for anvil clouds." *WMO Publications TD* (1995): 167-188.
[20] Kaufman, Y. J., et al. "Absorption of sunlight by dust as inferred from satellite and ground-based remote sensing." *Geophysical Research Letters* 28.8 (2001): 1479-1482.
[21] Nakanishi, Mikio, and Hiroshi Niino. "An improved Mellor–Yamada level-3 model with condensation physics: Its design and verification." *Boundary-layer meteorology* 112.1 (2004): 1-31.
[22] Beljaars, A. C. M., and A. A. M. Holtslag. "Flux parameterization over land surfaces for atmospheric models." *Journal of Applied Meteorology* 30.3 (1991): 327-341.

Hybrid ASUCA: Implementation Status



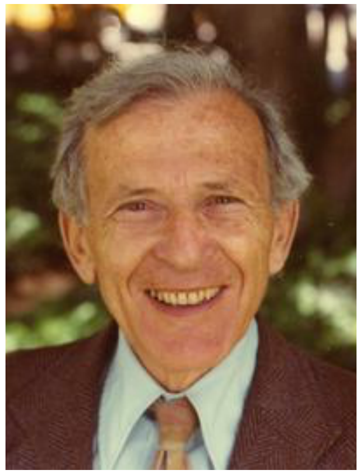
Numerical Weather Prediction (NWP)

Vilhelm Bjerknes
Image: Bjerknes family, CC-BY-SA



Lewis Fry Richardson
Image: Public Domain

- Bjerknes first formalized weather prediction problem in 1904 [1].
- Lewis Fry Richardson first attempted numerical weather prediction during WW1 using human computers - unsuccessfully due to numerical instability [2][3].
- Courant, Friedrichs and Lewy provided breakthroughs in numerical stability analysis in 1928 [4].
- Charney formulated the first practical NWP model. Together with Fjörtoft and Von Neumann this model was adapted for automatic computers after WW2 [5].



Hans Lewy
Image: George M. Bergman, GFDL

Jule Charney
Image: © Nora Rosenbaum, 1976

John von Neumann
Image: Public Domain

[1] Bjerknes, Vilhelm. "Das Problem der Wettervorhersage, betrachtet vom Standpunkte der Mechanik und der Physik." *Meteor. Z.* 21 (1904): 1-7.
[2] Woolard, Edgar W. "LF Richardson on weather prediction by numerical process." *Monthly Weather Review* 50.2 (1922): 72-74.
[3] Lynch, Peter. "Richardson's forecast: What went wrong?" *NOAA NWP* 50 (2004).
[4] Courant, Richard, Kurt Friedrichs, and Hans Lewy. "Über die partiellen Differenzengleichungen der mathematischen Physik." *Mathematische annalen* 100.1 (1928): 32-74.
[5] Charney, Jules G., Ragnar Fjörtoft, and J. von Neumann. "Numerical integration of the barotropic vorticity equation." *Tellus* 2.4 (1950): 237-254.

NWP Models

Equations:

1. **Hydrodynamic** equations of motion in 3D
   - ➡ 3 equations, differential relations among velocity components, density, air pressure
2. **Mass continuity** of air and water
3. **State** equation for **ideal gases**
4. **Conservation of energy**

➡ 7 equations, 7 unknowns, thus solvable

Dynamically modelled phenomena in free atmosphere:

- **Advection**
- **Diffusion**
- **Gravity waves**
- **Coriolis** force / Rossby waves
- **Sound waves**
  - No meteorological relevance but relevant for **stable** solutions of **large scale**, **high-Mach-number** atmospheric flows.
  - Time-splitting schemes used to allow sound wave resolution in fully compressible models.

NWP Models

Approximations:

- **Spherical-geopotential** (G): Gravity without horizontal component
- **Shallow-atmosphere** (S): Gravitation constant with distance from surface
- **Hydrostatic** (H): Atmosphere horizontally compressible, vertically incompressible
  - ➡ Sound waves filtered

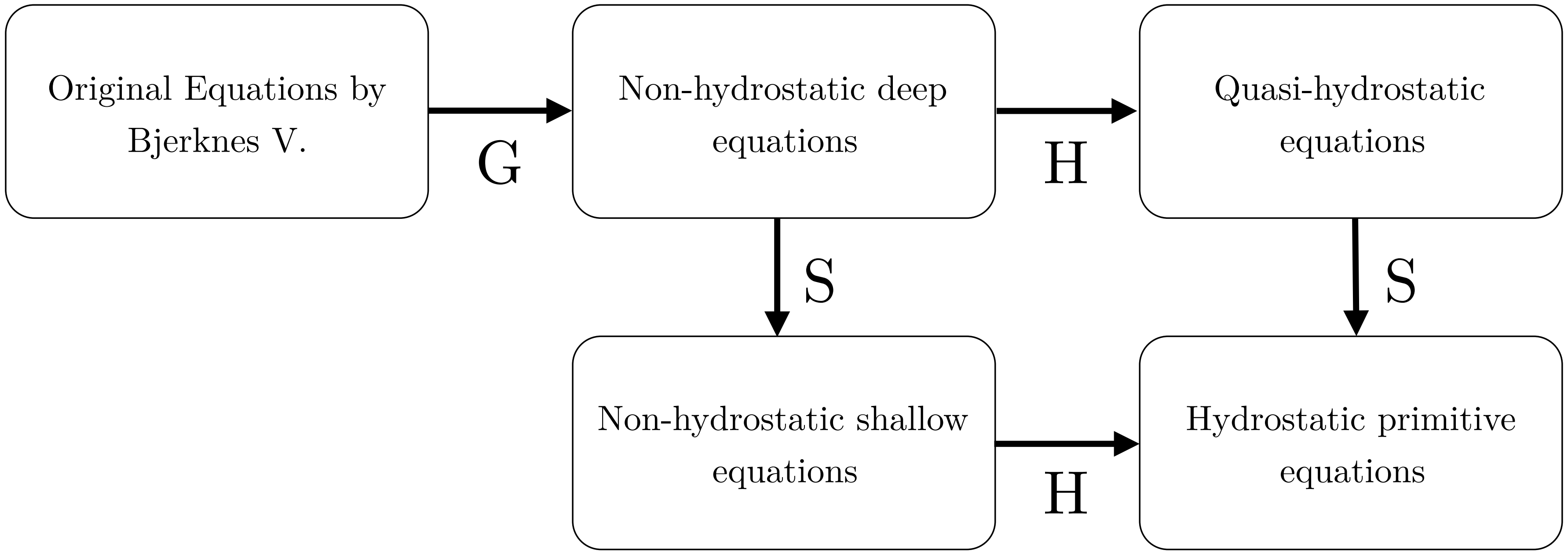


Figure 1.1: Interrelations of atmospheric models with respect to their approximations according to White et al.

ASUCA NWP Model

What is ASUCA**?**

- ``**A**suca is a **S**ystem based on a **U**nified **C**oncept for **A**tmosphere''
- **fully compressible**, **non-hydrostatic** weather prediction mode
- **regional scale** - as depicted in Figure 1.2
- one of **main operational** forecast models in Japan, **in production** since 2014
- spatial discretization: **finite-volume method** on **Arakawa-C-type rectangular** grid
- time discretization:
    - **third-order Runge-Kutta** based iteration scheme for **advection and Coriolis** force
    - **time-splitting method**, employing secondary third-order Runge-Kutta iteration with **short time step** for **sound-** and **gravity waves**
- **vertical-only** models for parametrization of **radiation**, **planetary boundary layer** and **surface physical processes**

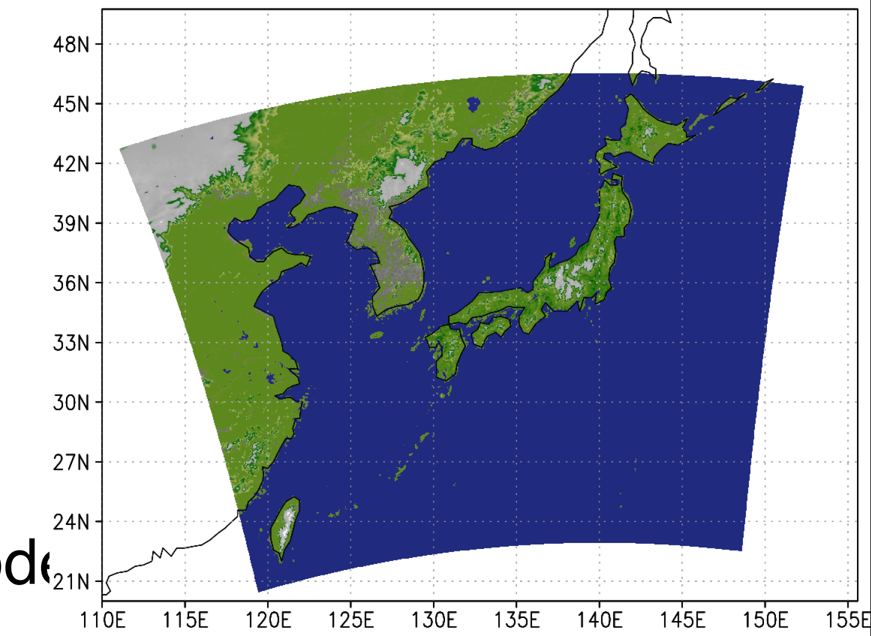


Figure 1.2: ASUCA's model simulation boundaries