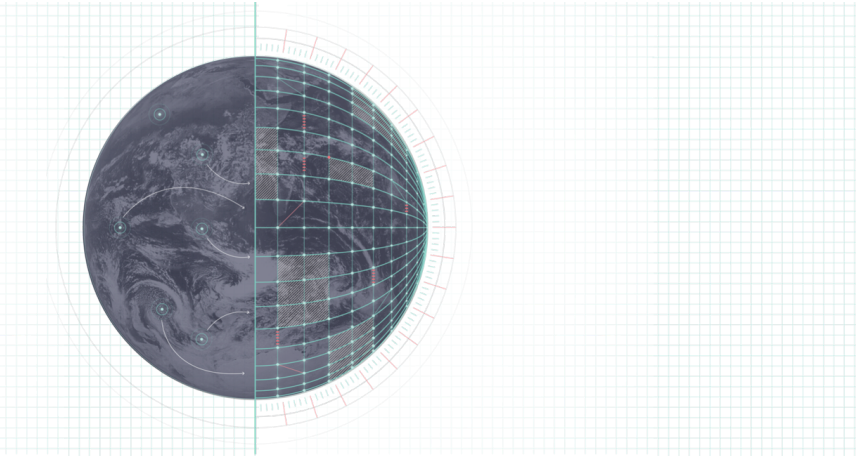
GPU ADAPTATION OF NWP SINGLE COLUMN ALGORITHMS

M. Staneker

A. Nawab, M. Lange, B. Reuter

Michael.Staneker@ecmwf.int
European Centre for Medium-Range Weather Forecasts

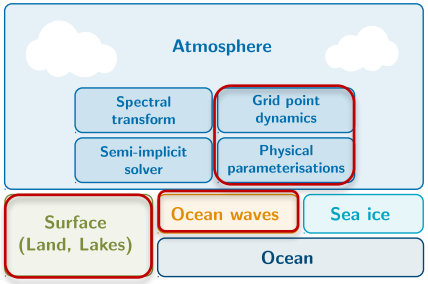


Funded by the European Union
Destination Earth
implemented by
ECMWF
esa
EUMETSAT

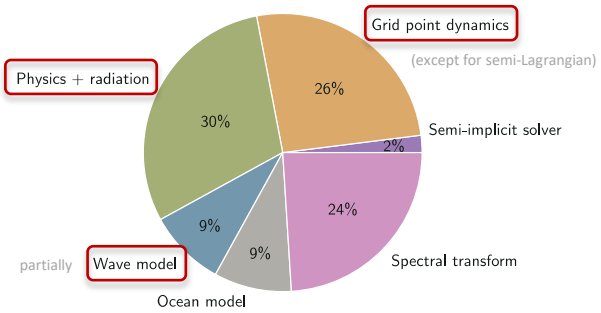
SINGLE COLUMN ALGORITHMS

Integrated **F**orecasting **S**ystem

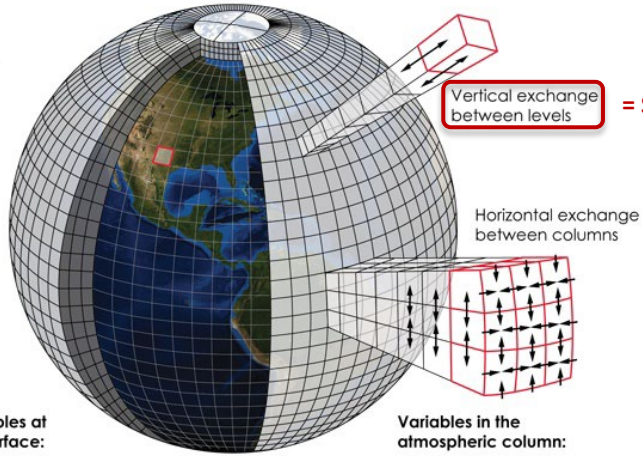
(at least partially) single column



Runtime shares at 9km horizontal resolution (operational HRES)



+ chemistry/atmospheric composition



= Single Column

Variables at the surface:

- Temperature
- Humidity
- Pressure
- Moisture fluxes
- Heat fluxes
- Radiation fluxes

Variables in the atmospheric column:

- Wind vectors
- Humidity
- Clouds
- Temperature
- Height
- Precipitation
- Aerosols

[1] **Valentin Clement et al.** "The CLAW DSL: Abstractions for Performance Portable Weather and Climate Models". In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. PASC '18. 2018. ISBN: 9781450358910. DOI: 10.1145/3218176.3218226.

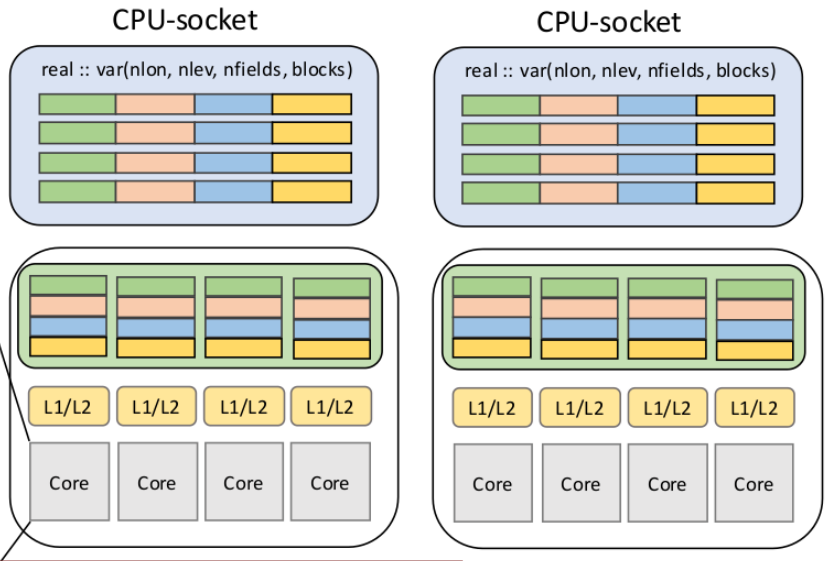
IFS: MEMORY DATA LAYOUT AND CPU PARALLELISATION

- Typical **data allocation**: (NPROMA, [NLEV], [NCLV], NGPBLKS)
  - Linearized horizontal dimension, allocated and iterated in blocks of size NPROMA
  - Optional vertical dimension and field count (species/variables) for some fields

```
!$omp parallel loop
DO ibl=1,nblocks
 CALL kernel(var1(:,:,ibl), var2(:,ibl), ...)
END DO
```

```
SUBROUTINE kernel (klon, klev, var1, var2, ...)

 INTEGER, INTENT(IN) :: klon, klev
 REAL, INTENT(INOUT) :: var1(klon, klev)
 REAL, INTENT(INOUT) :: var2(klon)

 DO j=1,klon
  var1(j, 1) = var2(j)
 END DO

 DO k=2,klev

  DO j=1,klon
   var1(j, k) = var1(j, k-1) + <update>
  END DO
 END DO
 ...

 END DO
END SUBROUTINE
```

SIMD



- **Typical NPROMA**: 16 (DP), 32 (SP)
- **Good vectorization** rate on current CPUs
- **Long** kernel: 2000+ LOC
- Many temporaries, **high register pressure**

GPU

GPU hardware characteristics: Lots of parallelism and very fast memory

- **Data needs to be moved to separate GPU memory space**

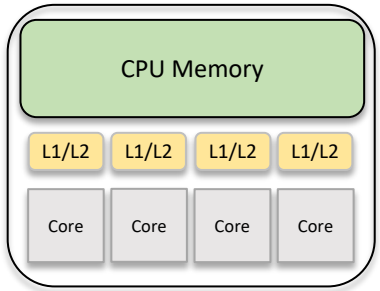
IFS-style vectorisation no longer suitable!

- GPU cores are grouped into **warps/wavefronts/vector thread**
  - **SIMT** extends **SIMD** from using Execution Units/Vector units to leverage threads …

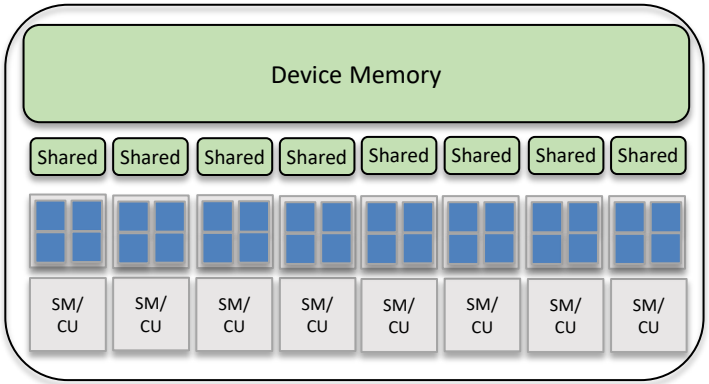
GPU compiler toolchains are crucial

- **NVIDIA** and **AMD**; **Intel** soon…

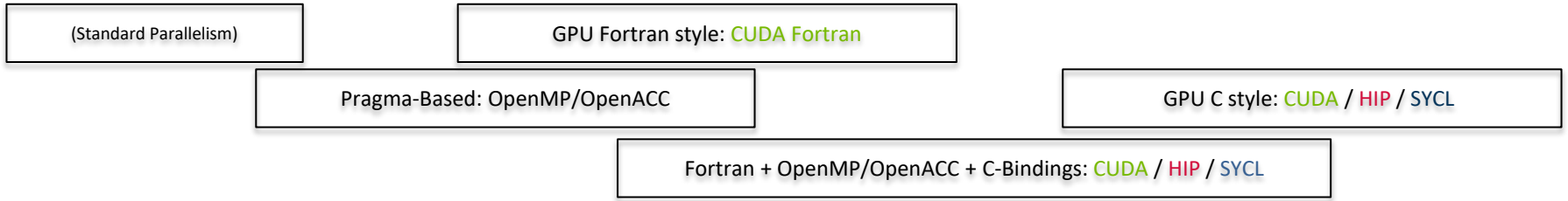
CPU



GPU = co-processor



In terms of language … starting from Fortran



Increasing amount of code changes necessary …

… but also offering better/more performance (optimisations)

In terms of concepts/layouts/optimisation strategies ….

increase parallelism | improve memory coalescing | improve data locality | cache usage | reduce register pressure | …

leading to different versions …

"CPU layout on device" (SIMD)

SCC - **S**ingle **C**olumn **C**oalesced (SIMT)

SCC-HOIST

SCC-STACK

SCC-K-CACHING

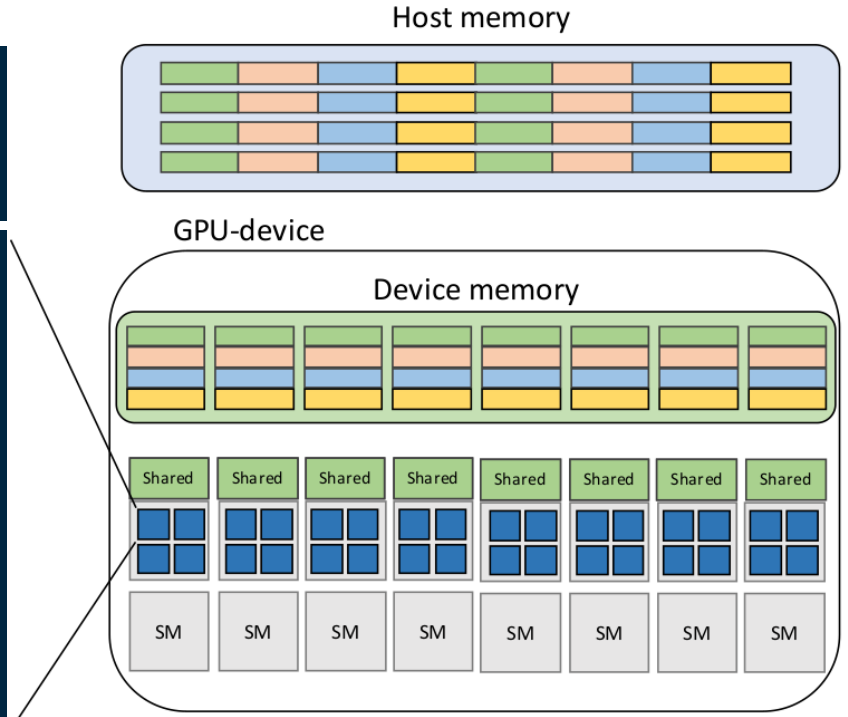
IFS: MEMORY DATA LAYOUT AND GPU PARALLELISATION

- Typical **data allocation**: (NPROMA, [NLEV], [NCLV], NGPBLKS)

```
DO ibl=1,nblocks
 CALL kernel(var1(:,:,ibl), var2(:,ibl), ...)
END DO
```

```
SUBROUTINE kernel (klon, klev, var1, var2, ...)

 INTEGER, INTENT(IN) :: klon, klev
 REAL, INTENT(INOUT) :: var1(klon, klev)
 REAL, INTENT(INOUT) :: var2(klon)

 DO j=1,klon

 var1(j, 1) = var2(j)

 DO k=2,klev
  var1(j, k) = var1(j, k-1) + <update>
 END DO
 ...

 END DO
END SUBROUTINE
```

SIMT



SCC VIA PRAGMA OFFLOAD

```
!$acc data copy(var1, var2, var3) !$omp target data map(tofrom: var1, var2, var3)

!$acc parallel loop gang vector_length(NPROMA) !$omp target teams loop bind(teams) ! or: !$omp target teams distribute
DO ibl=1,ngpblks
  CALL kernel(nproma, nlev, var1(:,:,ibl), var2(:,ibl), var3(:,:,1:3,ibl), ...)
END DO
```

```
SUBROUTINE kernel (klon, klev, var1, var2, var3, ...)
  INTEGER, INTENT(IN) :: klon, klev
  REAL, INTENT(INOUT) :: var1(klon, klev), var2(klon), var3(klon, klev, 3), ...
  REAL                :: tmp1, tmp2(klon, klev) ...
  !$acc routine vector !$omp declare target(kernel)

  !$acc loop vector !$omp loop bind(parallel) ! or: !$omp parallel do
  DO j=1,klon
    var1(j, 1) = var2(j)

    !$acc loop seq
    DO k=2,klev
      var1(j, k) = var1(j, k-1) + tmp1 + tmp2(j, k) + <update>
    END DO

  END DO
END SUBROUTINE
```

SCC VIA CUDA FORTRAN

```
REAL, DEVICE, ALLOCATABLE :: d_var1(:,:,:)
ALLOCATE(d_var1(nproma, nlev, ngpblks))
d_var1 = var1 ! copy host to device

GRIDDIM = DIM3(1, 1, ngptot/nproma)
BLOCKDIM = DIM3(nproma, 1, 1)

CALL kernel<<<GRIDDIM,BLOCKDIM>>>(nproma, nlev, ngpblks, var1, var2, var3, ...)
```

```
ATTRIBUTES(GLOBAL) SUBROUTINE kernel (klon, klev, kgpblks, var1, var2, var3, ...)
  INTEGER, INTENT(IN) :: klon, klev, kgpblks
  REAL, INTENT(INOUT) :: var1(klon, klev, kgpblks), var2(klon, kgpblks), var3(klon, klev, 3, kgpblks), ...
  REAL                :: tmp1, ...
  REAL, DEVICE        :: tmp2(klon, klev), ...

  j = THREADIDX%X
  ibl = BLOCKIDX%Z

  var1(j, 1, ibl) = var2(j, ibl)

  DO k=2,klev
    var1(j, k, ibl) = var1(j, k-1, ibl) + tmp1 + tmp2(j, k) + <update>
  END DO

END SUBROUTINE
```

SCC VIA CUDA/HIP/SYCL

CUDA **HIP** **SYCL**

- "gpuMalloc": *cudaMalloc* / *hipMalloc* / *sycl::malloc_device*
- "gpuMemcpy": *cudaMemcpy* / *hipMemcpy* / *sycl::queue::memcpy*
- Limit register pressure: *__launch_bounds__ and/*or *-maxrregistercount*

```
double *d_var1;

gpuMalloc(&d_var1, sizeof(double)*ngpblks*nlev*nproma);
gpuMemcpy(d_var1, var1, sizeof(double)*ngpblks*nlev*nproma, gpuMemcpyHostToDevice)

dim3 griddim(1, 1, ngptot/nproma);
dim3 blockdim(nproma, 1, 1);

kernel<<<griddim, blockdim>>>(nproma, nlev, ngpblks, var1, var2, var3, ...)
```

```
__global__ void kernel(int klon, int klev, int kgpblks, double * var, double * var2, double * var3, …) {
 double tmp1;
 double tmp2[klon*klev];

 jl = threadIdx.x;
 ibl = blockIdx.x;

 var1[j + klon*(1 + klev*ibl)] = var2[j + klev*ibl];

 for (k=2; k<klev; k+=1)
  var1[j + klon*(k + klev*ibl)] = var1[j + klon*((k-1) + klev*ibl)] + tmp1 + tmp2[j + klon*k] + <update>;
}
```

SCC VIA FORTRAN + PRAGMA DATA OFFLOAD + CUDA/HIP/SYCL KERNELS

Driver

.F90

```
!$acc data copyin(...) copy(var, ...) copyout(...) !$omp target data map(to: ...) map(tofrom: var, ...) map(from: ...)

CALL kernel_wrapper(nproma, nlev, ngpblks, var1, var2, var3, ...)
```

Fortran – C wrapper/interface

.F90

```
MODULE KERNEL_WRAPPER_MOD
 USE iso_c_binding
 INTERFACE
  SUBROUTINE kernel_launch(nproma, nlev, ngpblks, var1, var2, var3, ...) bind(c, name='kernel_launch')
   TYPE(c_ptr), value :: var
   INTEGER(C_INT)     :: klon, klev
  END SUBROUTINE
 END INTERFACE
 CONTAINS
  SUBROUTINE kernel_wrapper (nproma, nlev, ngpblks, var1, var2, var3, ...)
   INTEGER, INTENT(IN) :: klon, klev, kgpblks
   REAL, INTENT(INOUT) :: var1(klon, klev, kgpblks), ...
   !$acc host_data use_device(var1, ...) !$omp target data use_device_ptr(var1, ...)
   CALL kernel(klon, klev, ngpblks, c_loc(var1), ...)
  END SUBROUTINE
END MODULE
```

Kernel

.CXX

```
extern "C" {
 void kernel_launch(int klon, int klev, int kgpblks, double * var, double * var2, double * var3, ...) {
  kernel<<<griddim, blockdim>>>(klon, klev, var1, var2, var3, ...);
 }
}
__global__ void kernel(int klon, int klev, int kgpblks, double * var, double * var2, double * var3, ...) {
 double tmp1;
 double tmp2[klon*klev];

 jl = threadIdx.x;
 ibl = blockIdx.x;

 var1[j + klon*(1 + klev*ibl)] = var2[j + klev*ibl];

 for (k=2; k<klev; k+=1)
  var1[j + klon*(k + klev*ibl)] = var1[j + klon*((k-1) + klev*ibl)] + tmp1 + tmp2[j + klon*k] + <update>;
}
```

DEALING WITH TEMPORARIES

Motivation

```
DO ibl=1,ngpblks
  CALL kernel(..., var(:,:,ibl), ...)
END DO
```

```
SUBROUTINE kernel (klon, klev, var, ...)
  INTEGER, INTENT(IN) :: klon, klev
  REAL, INTENT(INOUT) :: var(klon, klev), ...
  REAL :: tmp1(klon), tmp2(klon, klev), ...
  ...
END SUBROUTINE
```

Bad on GPUs!

Better to have single allocation for all blocks ….

Idea / Possible solution(s):

HOIST – make temporaries to arguments and allocate on host

STACK – **pool allocator**: pass large chunk of memory allocated on host to device and assign temporaries

"k-caching" – get rid of/demote temporaries (if possible)

OPTIMIZED SIMT: SCC WITH HOISTING OF TEMPORARIES (SCC-HOIST)

HOIST – make temporaries to arguments and allocate on host

```
!$acc enter data create(tmp2) !$omp target enter data map(alloc: tmp2)
!$acc data copy(var1, var2, var3) !$omp target data map(tofrom: var1, var2, var3)

!$acc parallel loop gang vector_length(NPROMA) !$omp target teams loop bind(teams) ! or: !$omp target teams distribute

DO ibl=1,ngpblks
  !$acc loop vector !$omp loop bind(parallel) ! or: !$omp parallel do
  DO j=1,klon
    CALL kernel(nproma, nlev, var1(:,:,ibl), var2(:,ibl), var3(:,:,1:3,ibl), ..., j, tmp2(:,:,ibl))
  END DO
END DO
```

```
SUBROUTINE kernel (klon, klev, var1, var2, var3, ..., j, tmp2)
  INTEGER, INTENT(IN) :: klon, klev, j
  REAL, INTENT(INOUT) :: var1(klon, klev), var2(klon), var3(klon, klev, 3), ...,
  REAL, INTENT(INOUT) :: tmp2(klon, klev)
  REAL                :: tmp1, ...
  ...
END SUBROUTINE
```

OPTIMIZED SIMT: SCC WITH STACK (SCC-STACK)

STACK – **pool allocator:** pass large chunk of memory allocated on host to device and assign temporaries

```
ALLOCATE(scratch(large_num, ngpblks))

DO ibl=1,ngpblks
  stack_ptr = LOC(scratch(1, ibl))
  CALL kernel(..., var(:,:,ibl), ..., stack_ptr)
END DO
```

```
SUBROUTINE kernel (klon, klev, var, ..., stack_ptr)
  INTEGER, INTENT(IN) :: klon, klev
  REAL, INTENT(INOUT) :: var(klon, klev), ...
  REAL :: tmp1(klon), tmp2(klon, klev), ...
  POINTER(ptr_tmp1, tmp1)
  POINTER(ptr_tmp2, tmp2)
  ptr_tmp1 = stack_ptr
  stack_ptr = stack_ptr + 8 * SIZE(tmp1)
  ptr_tmp2 = stack_ptr
  stack_ptr = stack_ptr + 8 * SIZE(tmp2)
  ...
END SUBROUTINE
```

adapted from/analog to work at Météo-France

Extremely simplified!

K-CACHING: VERTICAL LOOP FUSION AND TEMPORARY DEMOTION

"k-caching" – get rid of/demote temporaries (if possible)

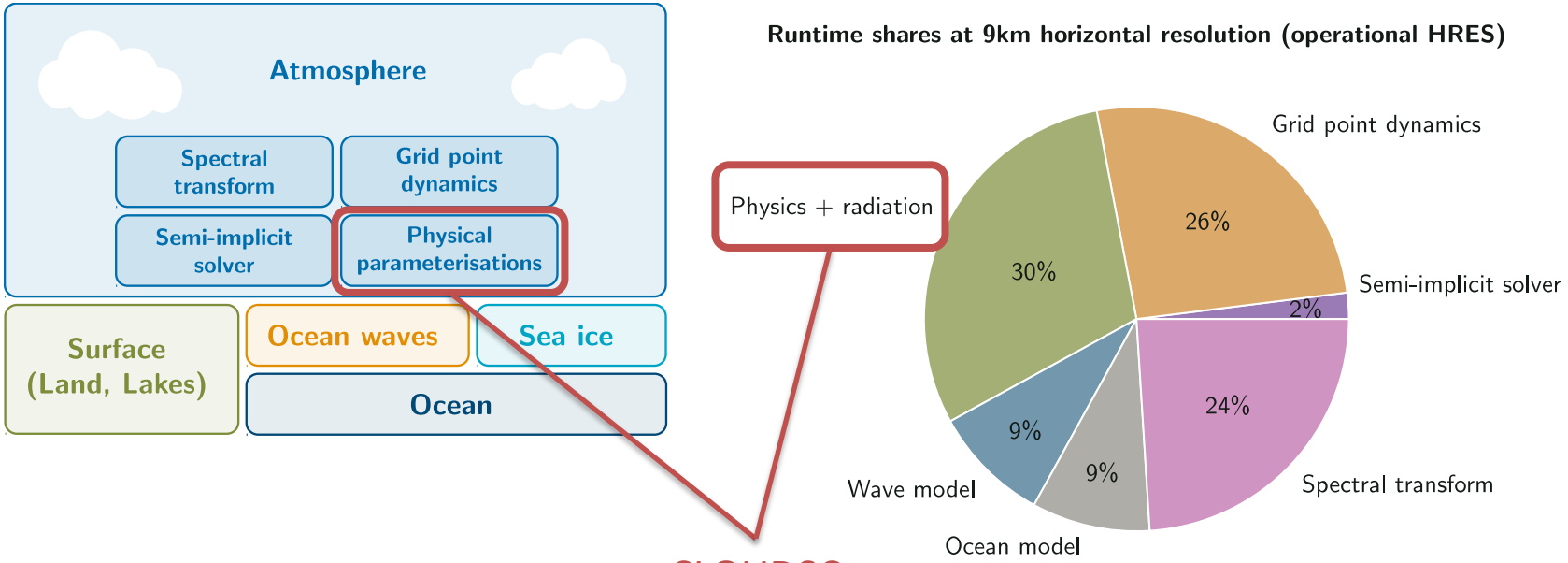
```
REAL :: Z(KLEV+2)
REAL :: X(KLEV)

DO K=1,KLEV+1
  Z(K) = K * constant
END DO

DO K=1,KLEV
  Z(K+1) = Z(K) + constant
END DO

DO K=10,KLEV
  Z(K) = Z(K) + constant
  X(K) = Z(K–1) + Z(K) = Z(K+1)
  …
END DO
```

```
REAL :: Z(3)
REAL :: X

DO K=1,KLEV+1
 K_I = MOD(K+1, 2) + 1
 K_IP1 = MOD(K+2, 2) + 1
 K_IM1 = MOD(K, 2) + 1

 IF (1<=K .AND. K<=KLEV+1) THEN
  Z(K_I) = K * constant
 END IF

 IF (1<=K .AND. K<=KLEV) THEN
  Z(K_IP1) = Z(K_I) + constant
 END IF

 IF (10<=K .AND. K<=KLEV) THEN
  Z(K_I) = Z(K_I) + constant
  X = Z(K_IM1) + Z(K_I) = Z(K_IP1)
  …
 END IF

END DO
```

- Fuse sequence of vertical loops
- **Loop-carried dependency only +/- 1**
- Reduced **register pressure** and **memory traffic**

Extremely simplified!

EXAMPLE: CLOUDSC MINI-APP/DWARF



CLOUDSC

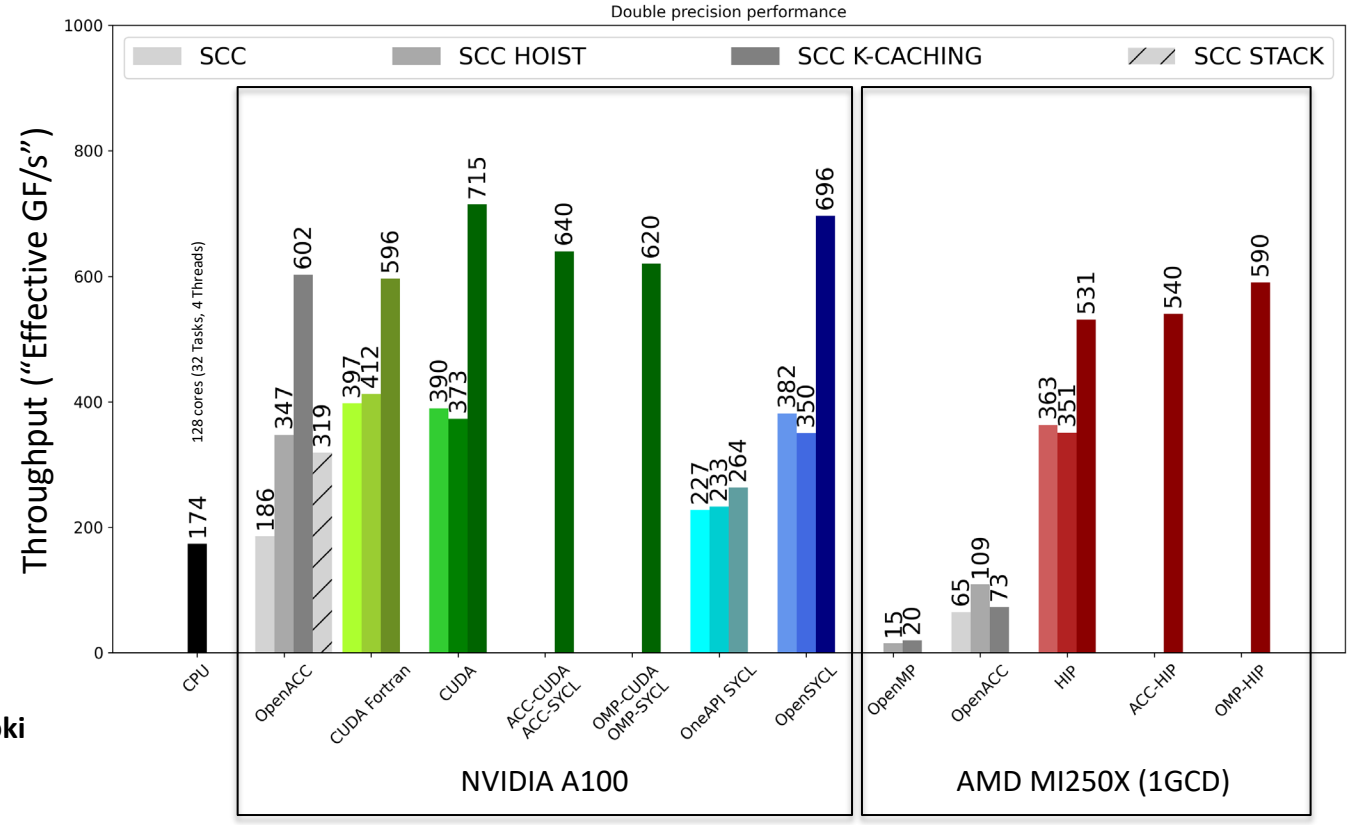
ca. 10% of operational forecast runtime

Representative for class of single-column algorithms

EXAMPLE: CLOUDSC MINI-APP/DWARF

CLOUDSC MINI-APP/DWARF

- Explore
  - Languages
  - Strategies
  - …
- Performance portability
- Automate
  Source-to-Source translation: **Loki**



/ecmwf-ifs/dwarf-p-cloudsc

- **Ahmad Nawab's talk:**
  "Automating GPU adaptation of NWP single column physics using Loki"
- **Balthasar Reuter's poster:**
  "Loki: A Source-to-Source Translation Tool for Numerical Weather Prediction codes and more"

ANY QUESTIONS?

Contact

Michael.Staneker@ecmwf.int
European Centre for Medium-Range Weather Forecasts

The work presented in this paper has been produced in the context of the European Union's Destination Earth Initiative and relates to tasks entrusted by the European Union to the European Centre for Medium-Range Weather Forecasts implementing part of this Initiative with funding by the European Union. Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

	GW-DRAG	Turbulence	Convection	Cloud
Fused	**0.87**	**9.41**	**12.57**	**14.22**
Split	3.76	15.93	17.41	20.59
GPU(SCC)	3.59	58.61	11.15	13.89
GPU(Stack) v0	**3.13**	**7.25**	**17.26**	**7.51**
GPU(Stack) v1	1.74	2.42	15.37	4.02
GPU(Stack) v2	**1.58**	**2.36**	**15.27**	**1.84**
GPU(Stack) v3	1.57	2.29	16.19	1.92

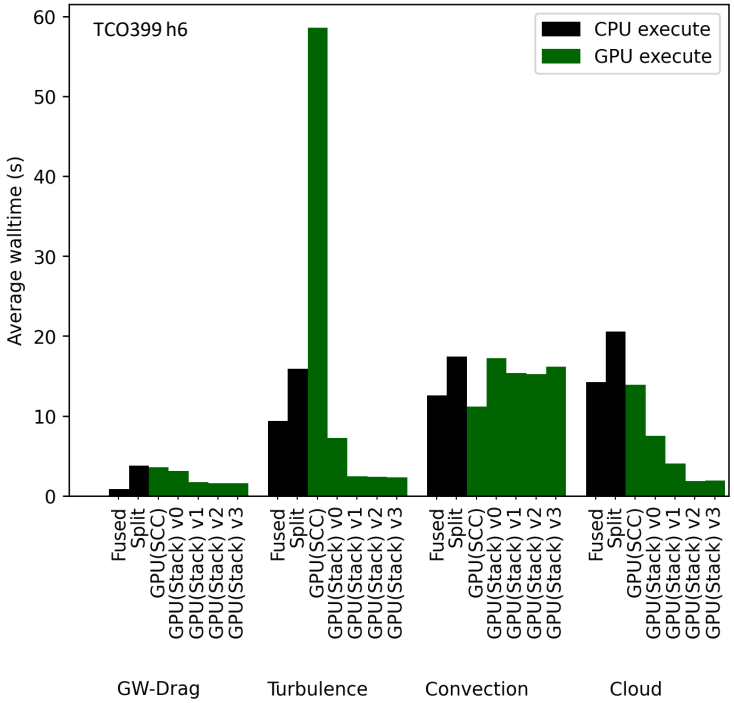
CPU baseline

GPU baseline: SCC

GPU SCC-Stack

↓ further work … tbc

! neglecting data movement !



- GPU(Stack)
  - v0: SCC-Stack
  - v1: + increased parallelism at driver level
  - v2: + manually inlined "pseudo-sequential" routine for CLOUD
    - **to be automated soon: (probably) very important for CONVECTION!**
  - v3: + maxrregistercount:128