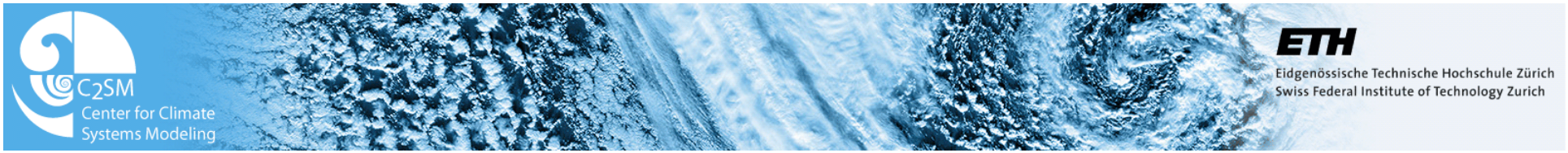


Are OpenACC directives the easy way to port Numerical Weather Prediction applications to GPUs ?

ECMWF HPC workshop 2014

X. Lapillonne, O. Fuhrer, S. Ruedhisuehli, A. Roches and the COSMO-HP2C team

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

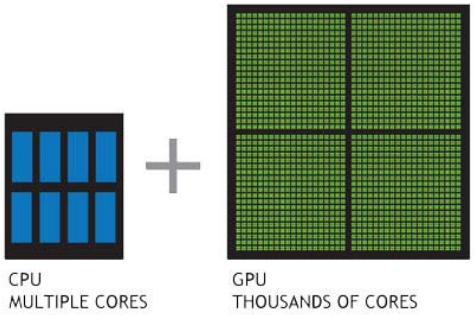
Eidgenössisches Departement des Innern EDI
Bundesamt für Meteorologie und Klimatologie MeteoSchweiz

CSCS
Swiss National Supercomputing Centre

GPU computing : some key aspects

- Hybrid approach : offload parts of an application, while the remainder stays on the CPU

- GPUs have thousands of compute cores : need to express fine grain parallelism



- GPU and CPU have (currently) separate physical memory
  - requires specific data management
  - data transfer may be a performance issue (slow transfer via PCI bus)

- As compared to typical multicore CPUs, GPUs have :
  - 7x higher peak performance[1] (double precision)
  - 4x higher memory bandwidth
  - Equivalent energy consumption

[1] Intel Sandy Bridge CPU vs Nvidia K20 GPU

OpenACC compiler directives

OpenACC : Open standard, supported by 2 compiler vendors PGI, Cray

```
!$acc data copyout(a), copyin(b,c)
!$acc parallel
!$acc loop gang vector
do i=1,N
    a(i)=b(i)+c(i)
end do
!$acc end parallel
!$acc end data
```

Controls data location (GPU/CPU)

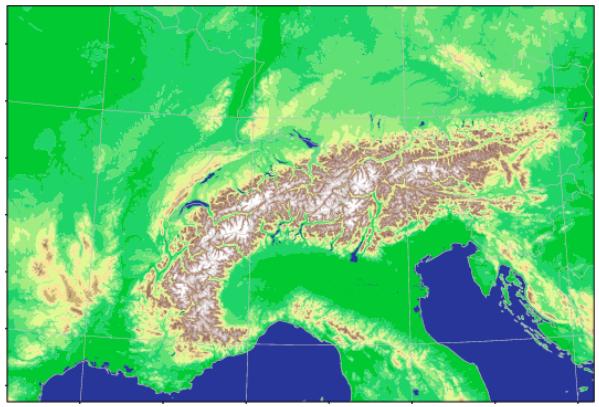
Tells the compiler to generate GPU code

Controls parallel execution

Directives vs GPU-specific language

	GPU-Specific language (Ex: CUDA)	Compiler directives
Implementation, porting effort	Re-write	Incremental adaptation of existing code
Control over parallel execution	Yes	Yes
Control over memory placement (CPU or GPU)	Yes	Yes
Control over local memory hierarchy (shared, texture …)	Yes	No (automatic)
Software management	Often separate GPU and CPU code	Single source possible
Target Architecture	CUDA: Nvidia, OpenCL: multiple	Multiple

The COSMO-GPU project

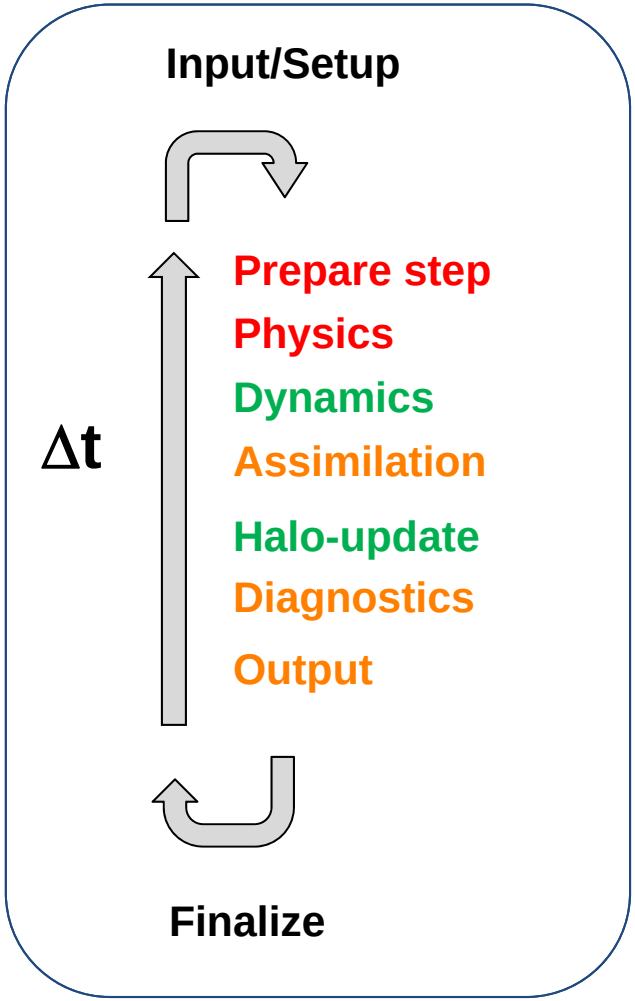


- COSMO : Limited-area model, structured grid
  - Run operationally by 7 national weather services
    Germany, Switzerland, Italy, Poland, …
  - Used for climate research in many academics institutions : ETHZ, …
- Part of a larger project (HP2C initiative):
  - runs efficiently on different architectures : Multicore CPUs (x86) and GPUs
  - allow domain scientist to easily bring new developments
  - will still be a good fit in the future
- Porting strategy:
  - Low compute intensity => too costly to transfer data for selected compute intensive parts
  - Full port strategy [1]

[1] Fuhrer, O. et al., Supercomputing Frontiers and Innovations, 1, 2014

Our approach : full GPU port

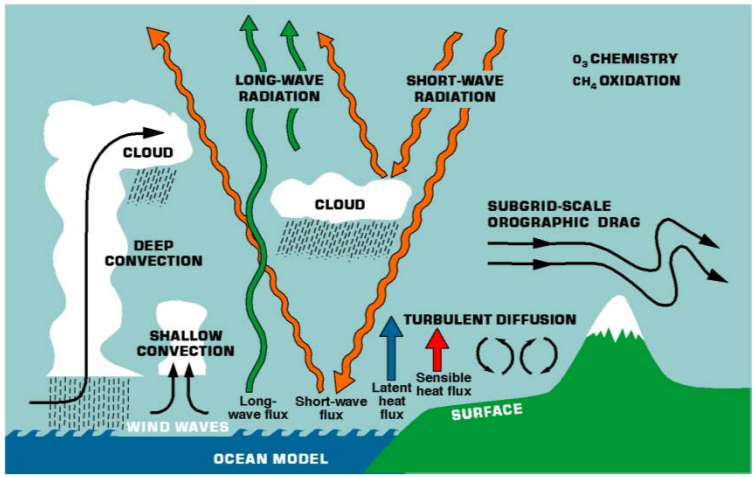
- **Avoid CPU-GPU copies by executing all the time loop computation on GPU**



→ keep on CPU / copy to GPU

→ **OpenACC directives**
→ **OpenACC directives**
→ **STELLA Library**
→ **mixed CPU and GPU (OpenACC)**
→ **GPU-GPU communication library (GCL)**
→ **mixed CPU and GPU (OpenACC)**
→ **mixed CPU and GPU (OpenACC)**

Physics implementation



- Parameterizations are a time critical components of the model :
  => Performance requirements

- Code optimized for GPU [1]:
  - loop restructuring (reduce kernel overhead, improve reuse)
  - scalar replacements
  - on the fly computations (reduce memory accesses)
  - manual caching (using scalars)
  - replacement of local automatic arrays with global automatic arrays (avoid frequent memory allocation on the GPU)

- Some GPU optimizations degrade performance on CPU : keep separate routines when required (using ifdef)

[1] Lapillonne and Fuhrer, Parallel Processing Letters, 24, 2014

Different optimization requirements in the physics on CPUs and GPUs

Profiler information in the physics:

- **CPU :** compute bound
  - Compiler auto-vectorization : easier with small loop construct
  - Pre-computation

- **GPU :** memory bound limited
  - Benefit from large kernels : reduce kernel launch overhead, better computation/memory access overlap
  - Loop re-ordering and scalar replacement
  - On the fly computation

Code example

Original code

```
do k=2,nk
 do i=1,ni
    some code 1 …
    c(i) = D*exp(a(i,k-1))
 end do
 do i=1,ni
    a(i,k)=c(i)*a(i,k)
    some code 2 …
 end do
end do
```

OpenACC 1

```
do k=2,nk
 !$acc parallel
!$acc loop gang vector
 do i=1,ni
    some code 1 …
    c(i) = D*exp(a(i,k-1))
 end do
!$acc end parallel
!$acc parallel
!$acc loop gang vector
 do i=1,ni
    a(i,k)=c(i)*a(i,k)
    some code 2 …
 end do
!$acc end parallel
end do
```

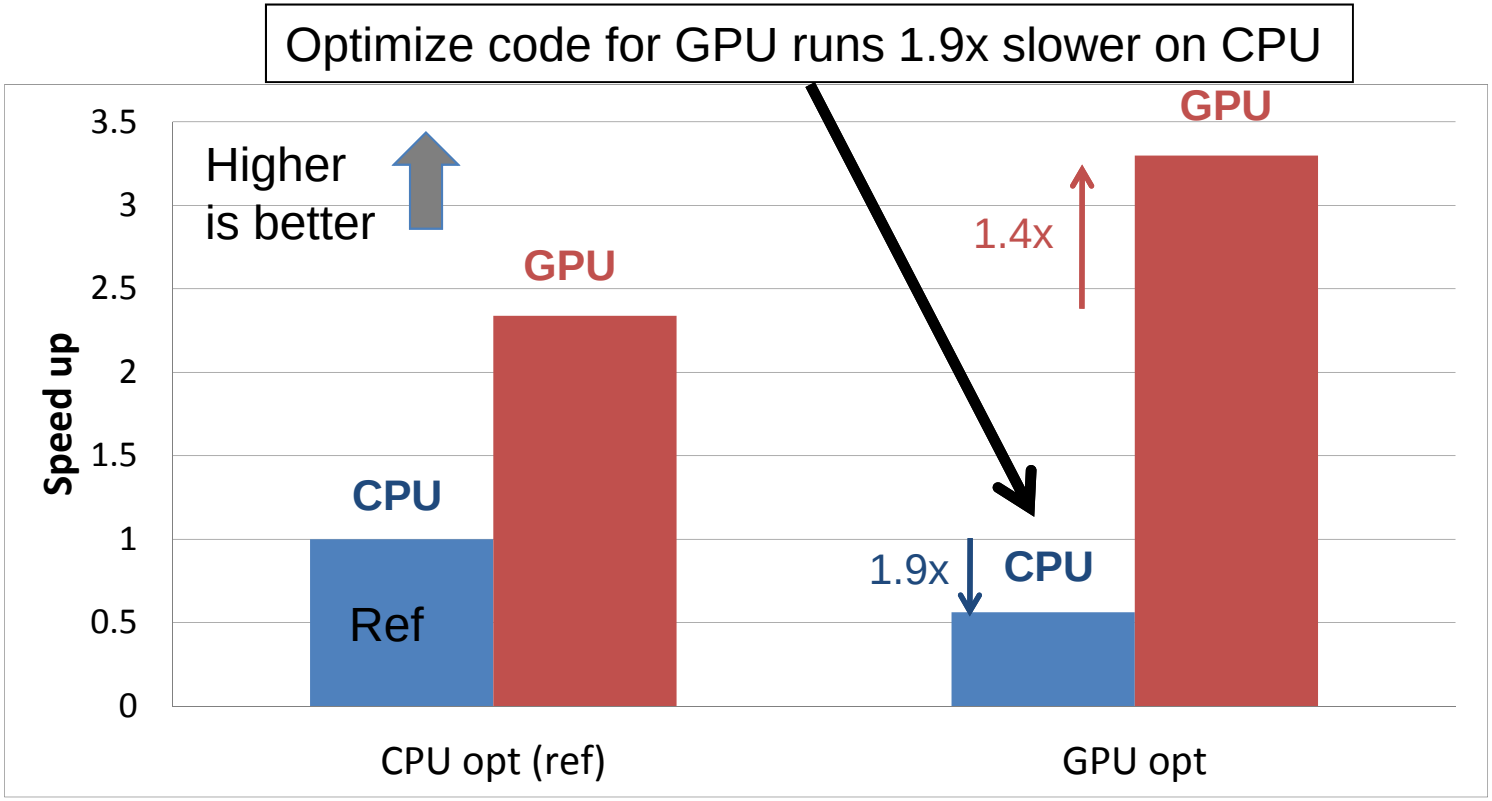
OpenACC 2

```
 !$acc parallel
  !$acc loop gang vector
  do k=2,nk
   do i=1,ni
     some code 1 …
     zc=D*exp(a(I,k-1))
     a(I,k)=c(i)*a(I,k)
     some code 2 …
    end do
  end do
  !$acc end parallel
```

OpenACC 1 : Keep performance on CPU, not optimal on CPU
OpenACC 2 : Optimal performance on GPU, may decrese perfromance on CPU

Example in the radiation

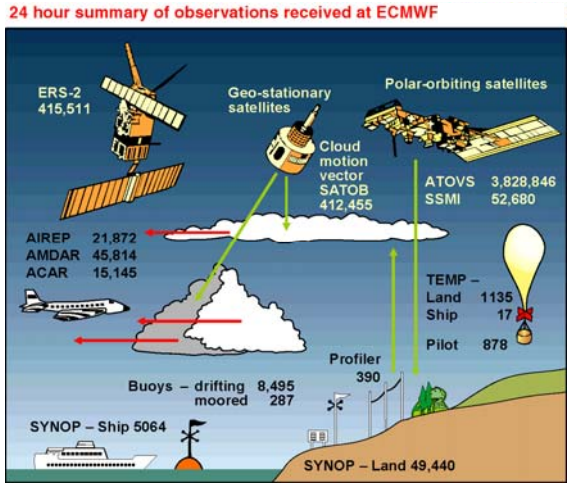
- Considering key kernel of radiation
- Test domain 128x128x60, 1 Sandy Bridge CPU vs 1 K20x GPU



Speed up with respect to reference code on CPU

Radiation is the strongest example, on average in the physics, code optimized for GPU is 1.3x times slower when run on CPU

Porting assimilation using OpenACC

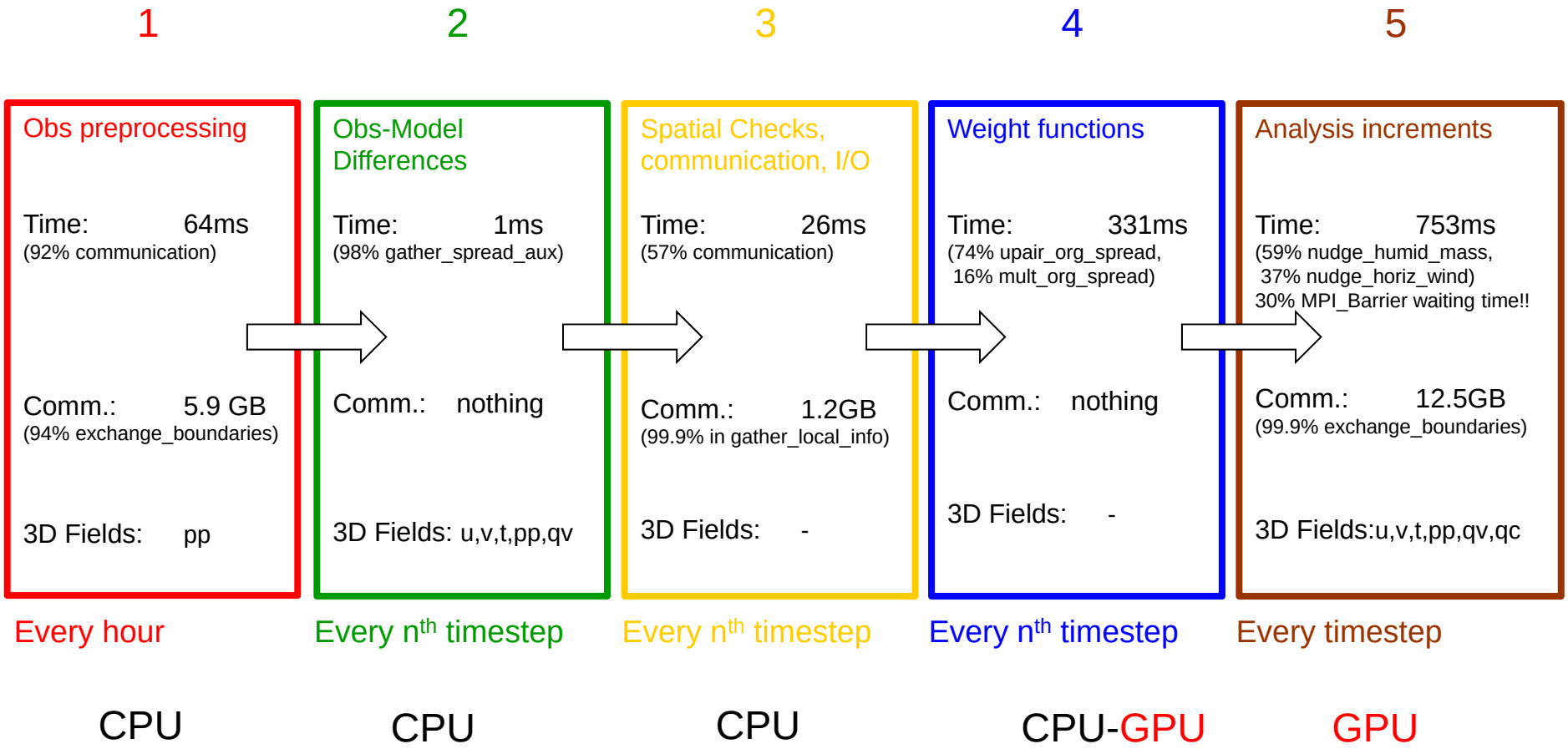


- Data assimilation is a large code (O(100K) lines),
  not performance critical:
  priority => code maintenance, keep single source code

- Aim: Achieve modest performance while keeping porting effort at a reasonable level
  - parts running on CPU
  - parts on GPU
  - minimize required data transfer

- Transformations / optimizations for GPU
  - adapt non parallel code
  - replacement of local automatic arrays with global automatic arrays (avoid frequent memory allocation on the GPU)

Assimilation porting strategy



Experience using OpenACC in the Assimilation

- Well adapted to port large part of the code
- Possible to keep parts on CPU parts on GPU
- Key to get acceptable performance :
  - manual data placement and transfer between CPU and GPU (minimize data transfer)
  - Main source of error missing CPU-GPU update, difficult to find.

Assimilation

In the assimilation some parts are run on the CPU other on the GPU. This requires careful data management

mult_org_spread :

```
   ! Update kviflml on GPU after it has been written on CPU.
   !$acc update device ( kviflml(:,:,:) )

 ENDIF    !  (k == ke)


!-------------------------------------------------------------------------------
!  Section 3: Organize the spreading of observation increments
!-------------------------------------------------------------------------------


! Get the horizontal correlation scales
! and the non-divergent correction factor to the 2-dim. wind correlations
! ---------------------------------------------------------------------

  !$acc parallel
  !$acc loop gang vector
  DO istaml = 1 , nmltot
    ilv (istaml) = 0
    ista = isortml(istaml)
    IF (lsprsml(ista)) THEN
      IF (      (k <= MAX( kviflml(ista,1,1) , kviflml(ista,3,1)
                         , kviflml(ista,1,2) , kviflml(ista,3,2)
```

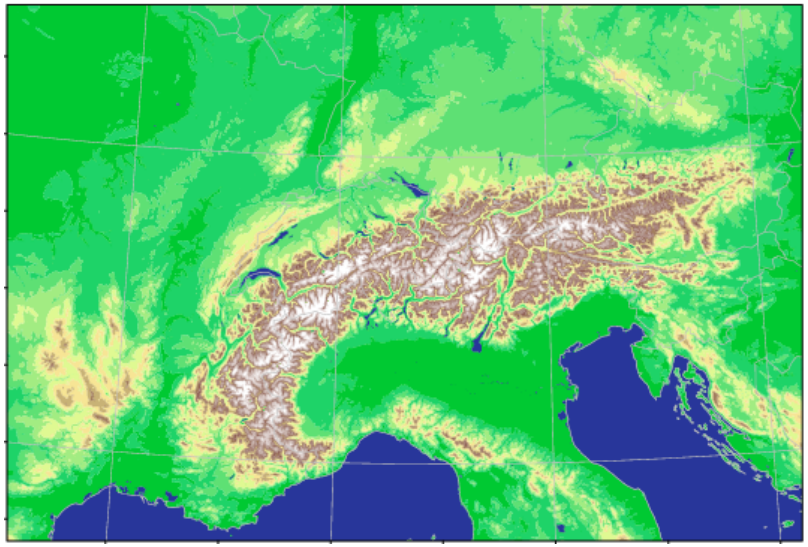
Previous part on CPU
Requires an update

Running on GPU

OpenACC effort in COSMO

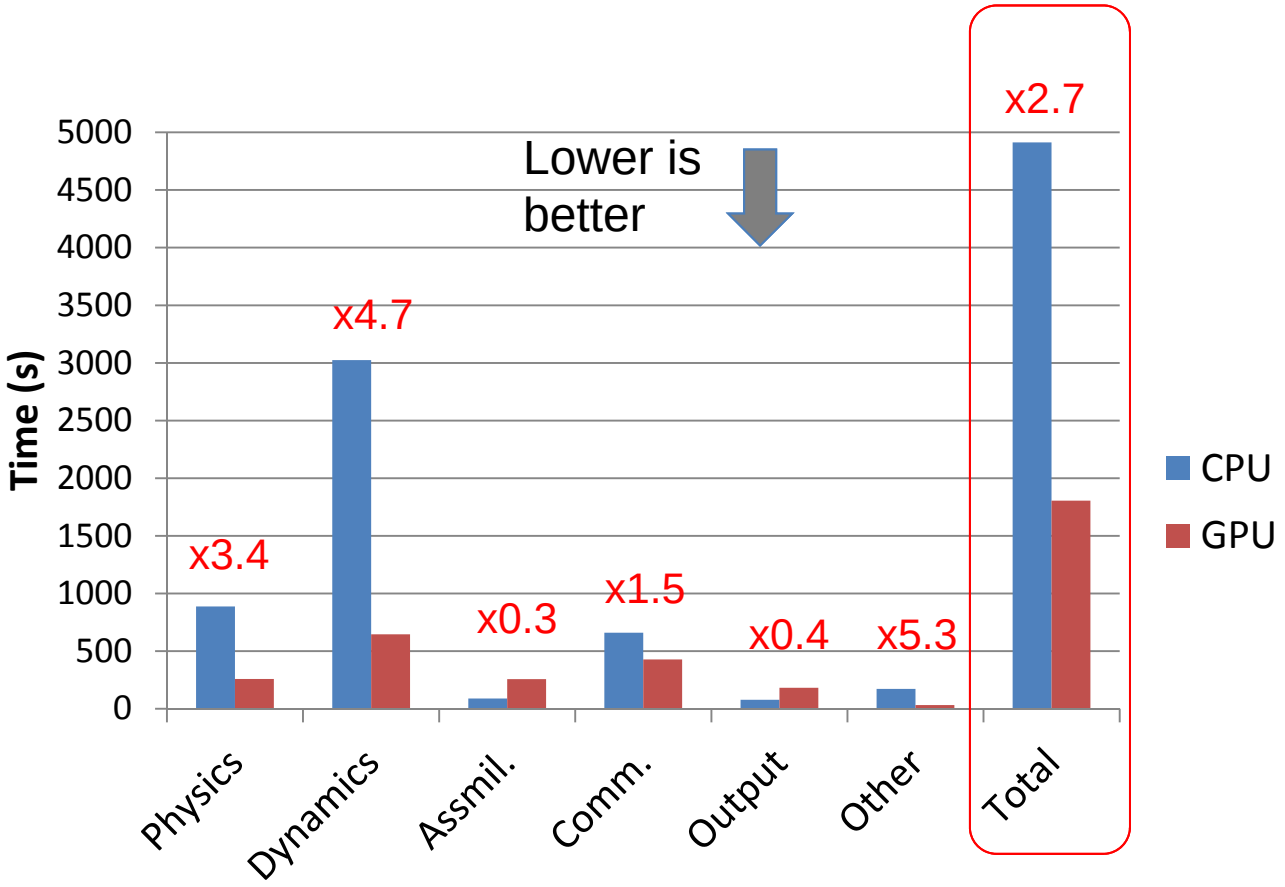
- > 4000 OpenACC statements
- Significant code restructuring : 40 source files out 160 modified
- Contribution from many developers:
  S. Ruedhisuehli, T. Diamanti, A. Roches, D. Leutwyler, C. Padrin , S. Schaffner.

Performance results



- COSMO-2 (2 km) Meteoswiss configuration
  24h run, 520x350x60 grid points
  - dynamics
  - physics : microphysics, radiation, turbulence, soil, shallow convection, sso
  - assimilation : nudging
  - output
- Socket to Socket comparison, using 9 compute nodes, on XK7 (todi) :
  - GPU nodes : AMD Opteron CPU (using 1 cores) + K20x GPU
  - CPU nodes : AMD Opteron CPU (using all 16 cores)

Performance results

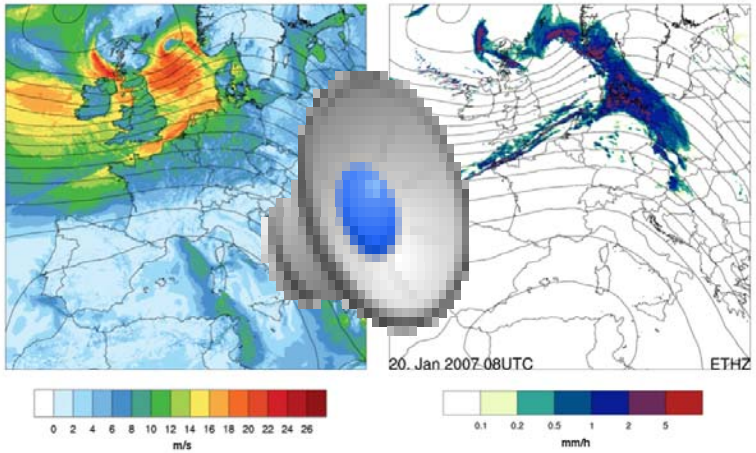


24h COSMO-2 Simulation, 9 K20x GPU vs 9 AMD Opteron CPU

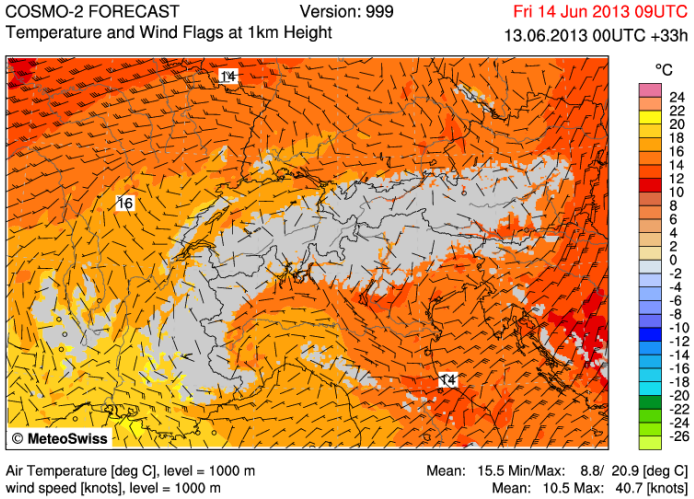
- Overall run 2.7x faster on GPU
- Physics, OpenACC, with optimizations : 3.4X faster on CPU
- Assimilation, OpenACC, parts on CPU – 1 core : 3.3X slower, < 15% of runtime

Ongoing work/applications

- Prototype GPU code used for cloud-resolving climate simulations at European-scale (D. Leutwyler, ETHZ)



Testcase : 7 days in Jan. 2007, includes winter storm Kyrill (18.01.2007), PiZ daint 144 nodes



33h simulation on prototype OPCODE system (8 GPUs).

- Prototype GPU code has been run daily for several months for validation
- Merge back changes into the official COSMO trunk (ongoing)
- New COSMO version will be used for developing the new Meteoswiss operational configuration : 1 km grid and ensemble forecast (starting in November 2014)

Conclusion : is it easy ?

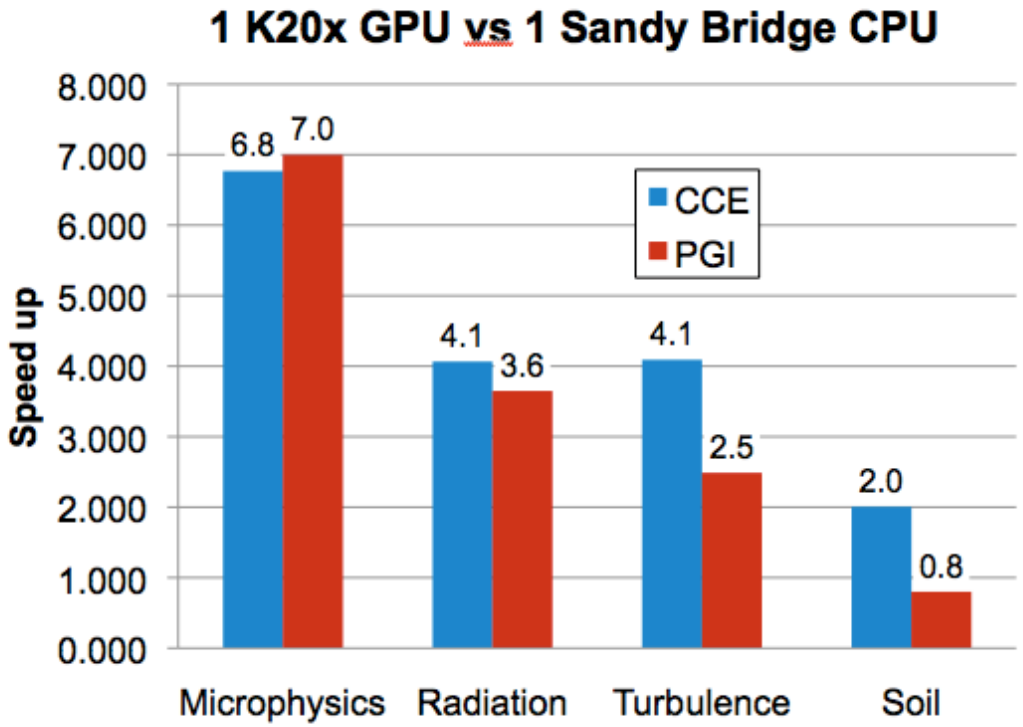
- OpenACC appropriate to port large part of the code, was key to our full port strategy. Overall code 2.7x faster on GPU than CPU.
- Simple port (no restructuring) useful to avoid data transfer, but usually leads to limited performance
- Manual data placement and transfer between CPU and GPU is key to get good results -> add some complexity
- It is possible to achieve good performance in the physics. This requires substantial code refactoring
- Some of the optimization for GPU degrade performance on other architecture : not always performance portable – Different code path used in some routine
- May need extension to OpenACC or new tools to abstract:
  - loop placement/order
  - promotion/demotion of local variables
  - caching

C2SM
Center for Climate
Systems Modeling

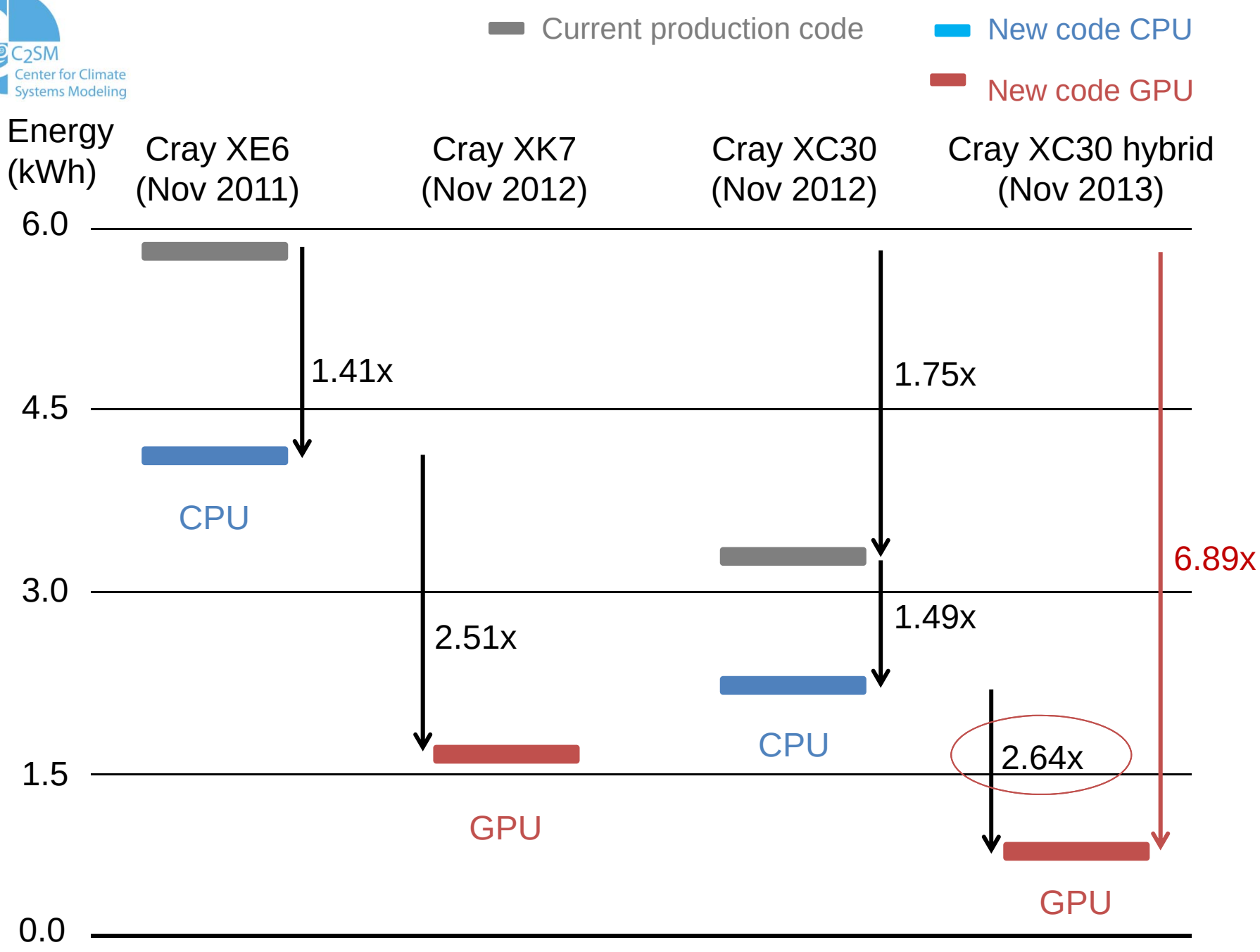
Thank you

Socket to socket comparison: Physics

- Test domain 128x128x60 8 core Intel Sandy Bridge CPU vs K20x GPU (using double precision)

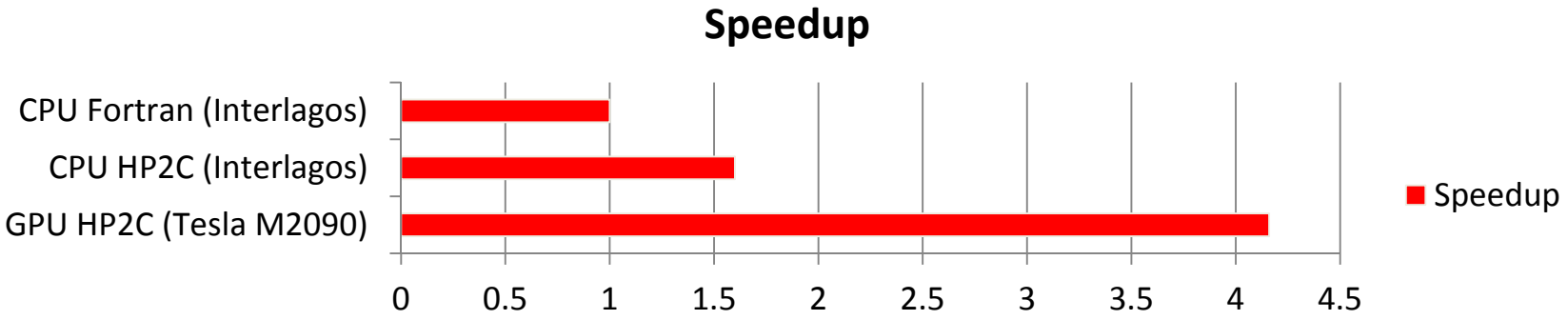


- 90% of physics run time on original CPU code: microphysics, radiation and turbulence -> Higher optimization effort


C2SM
Center for Climate
Systems Modeling
Current production code
New code CPU
New code GPU
Energy
(kWh)
Cray XE6
(Nov 2011)
Cray XK7
(Nov 2012)
Cray XC30
(Nov 2012)
Cray XC30 hybrid
(Nov 2013)
6.0
4.5
3.0
1.5
0.0
1.41x
1.75x
6.89x
2.51x
1.49x
2.64x
CPU
CPU
GPU
GPU

Socket to socket comparison: Dynamics

- **Test domain 128x128x60. CPU: 16 cores Interlagos CPU; GPU : X2090**



What does this mean for NWP application ?

- Low FLOP count per load/store (stencil computation)
- Example with COSMO-2 (operational configuration at MeteoSwiss) :

*

Part	Time/Δt
Dynamics	**172 ms**
Physics	**36 ms**
Total	**253 ms**

vs

§

Transfer of ten prognostic variables

118 ms

CPU-GPU data transfer time is large with respect to computation time:

Accelerator approach might not be optimal

* CPU measurements: Cray XT6, Magny-Cours, 45 nodes, COSMO-2

§ GPU measurements: PCIe 8x, HP SL390, 2 GB/s one way, 8 sub-domains