Towards a full GPU version of the COSMO model

Status of the COSMO priority project Performance on Massively Parallel Architectures (POMPA)

Oliver Fuhrer[1], Xavier Lapillonne[2], Tobias Gysi[3],
Carlos Osuna[2],Tiziano Diamanti[1], **Philippe Steiner**[1]

[1]MeteoSwiss Zurich, Switzerland
[2]C2SM/ ETH Zurich, Switzerland
[3]Supercomputing Systems AG, Switzerland

Outline

- COSMO Priority Project POMPA
- Why GPUs are attractive for COSMO
- Approach
- Results
- Outlook
- Conclusion

COSMO Priority Project POMPA

- **P**erformance **O**n **M**assively **P**arallel **A**rchitectures
- 4 year project (09.2010 – 09.2014)

- Projects HP2C COSMO & HP2C OPCODE of the initiative HP2C (High Performance & High Productivity Computing) funded by the Swiss Universities embedded
- HP2C finishes mid of 2013

- **Goal**
  Prepare the COSMO code for these future high performance computing (HPC) architectures

Supercomputers are changing!

- **Massive parallelism** – increase in number of cores, stagnant or decreasing clock frequency
- **Less and “slower” memory per thread** – memory bandwidth per instruction/second and thread will decrease, more complex memory hierarchies
- **Heterogeneous hardware** – mixed clusters of CPUs and accelerators (GPUs)
- **Only slow improvements of inter-processor and inter-thread communication** – interconnect bandwidth will improve only slowly
- **Stagnant I/O sub-systems** – technology for long-term data storage will stagnate compared to compute performance

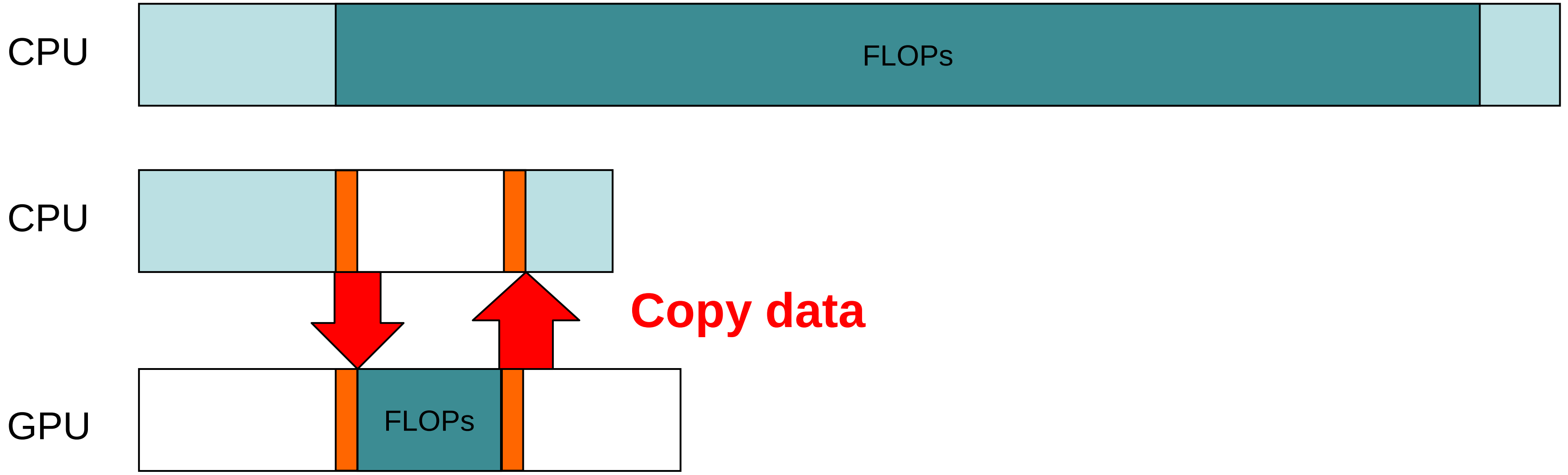
We need to adapt our codes in order be efficient in the future!

Potential of GPUs

Chip	CPU (Interlagos)	GPU (Tesla X2090)
		
Architecture	16 cores	512 cores
	compute intensive × 5	
Peak Performance	134 GFlops	665 GFlops
	memory intensive × 3	
Memory Bandwidth	51.2 GB/s	155 GB/s
Power Consumption	115 Watt	225 Watt
Price per Socket	~ X $	~ X $

Accelerator approach

- Leverage high peak performance of GPU
- CPU and GPU have different memories



- This approach does not work for COSMO!
  - Low FLOPs count per load/store (stencils!)
  - Transfer of data on each timestep too expensive

Why this doesn't work for COSMO

- Low FLOP count per load/store (stencils!)
- Transfer of data on each timestep too expensive

*

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

vs

§

Transfer of ten prognostic variables

118 ms

All code which touches the prognostic variables within timestep has to be ported

Full GPU Port

Common goal of the projects HP2C COSMO / HP2C OPCODE

GPU-implementation of “full” timestep of COSMO

Aim for…

- Completeness (i.e. full COSMO model)
- Performance (i.e. lower time-to-solution)
- Portability / Maintainability (i.e. no hacks)
- Durability (i.e. knowledge transfer and documentation)

•Time/resource constraints lead to compromises

Approach

Dynamical core

- Small group of developers
- Memory bandwidth bound
- Complex stencils (3D)
- 60% of runtime

→ **Complete rewrite in C++/CUDA**
→ Development of a stencil library
→ Development of new communication library (GCL)
→ Target architecture CPU (x86) and GPU.
→ Extendable to other architectures
→ Long term adaptation of the model

Physics and Data Assimilation

- Large group of developers
- Code may be shared with other models
- Less memory bandwidth bound
- Large part of code (50% of the lines)
- 20% of runtime

→ **GPU port with compiler directives (OpenACC)**
→ Little code optimization
→ Some parts stay on CPU
→ Most ported routines currently have CPU and GPU version

Performance of Dynamical Core

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

CPU Version

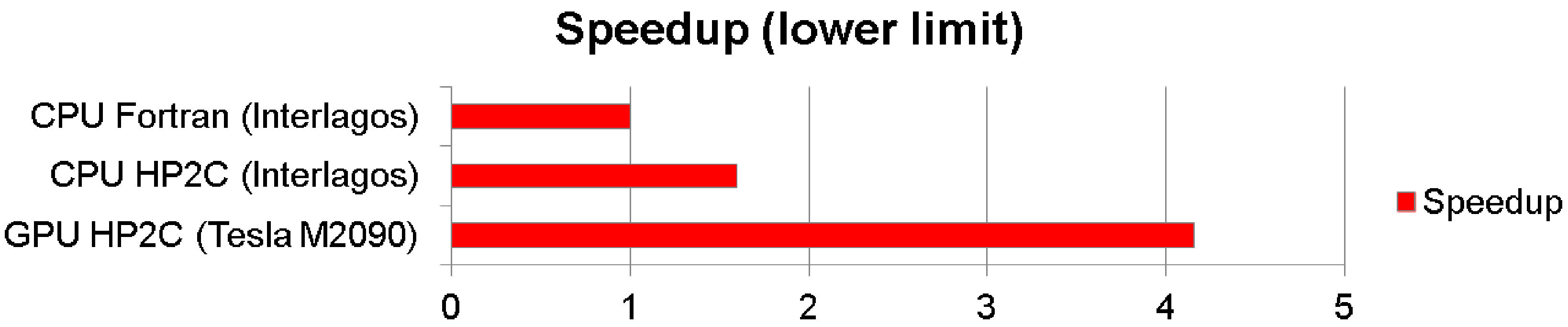
•Factor 1.6x – 1.8x faster than the COSMO dycore

•No explicit use of vector instructions (potential for 10-30% improvement)

GPU Version

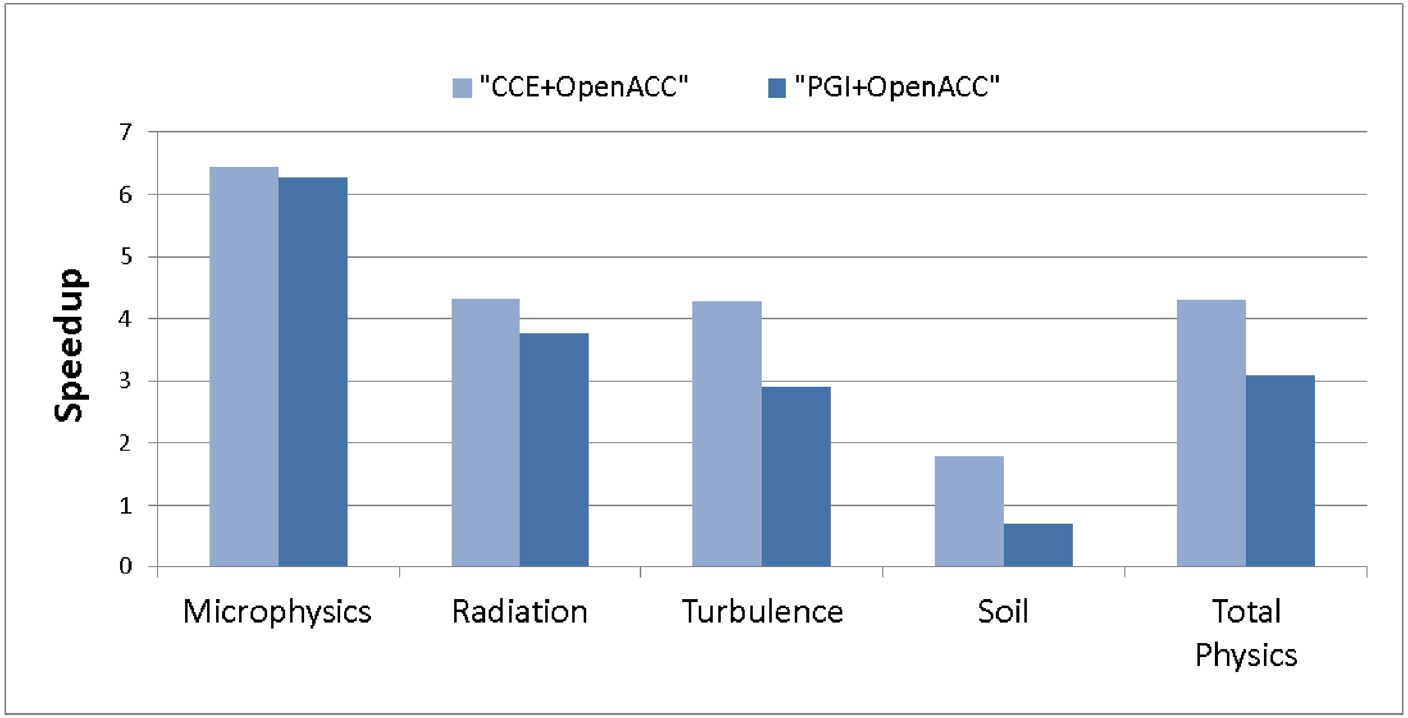
•Same generation GPU is roughly a factor 2.6x faster than CPU

•Potential for further performance optimizations



Performance of Physics

- Test domain 128x128x60 – 16 cores CPU vs GPU (Kepler)



- Overall speed up ~4x
- Similar performance with OpenACC (Cray and PGI compiler)
- Running the GPU-Optimized code on CPU is about 25% slower
  → separate source code for time critical routines

Current Status

Setup

Δt

Input	➡	keep on CPU / copy to GPU
Physics	➡	directives, (soil, radiation, microphysics, turbulence)
interface		
Dynamics	➡	C++/CUDA rewrite, (RK dynamical core)
interface		
Assimilation	➡	directives, (nudging)
Halo-update	➡	communication library (GCL), (halo, scatter/gather)
Diagnostics	➡	directives, only bare essentials
Output	➡	keep on CPU / copy from GPU

Cleanup

Ongoing tasks

- Porting of dynamical core
  - lateral boundary relaxation (Carlos Osuna, C2SM)
  - new fast waves solver (Michael Baldauf, DWD)
  - explicit horizontal diffusion (Master student, ETH)
  - semi-Lagrangian advection (Bachelor student, ETH)
- Porting of physical parametrizations
  - Tiedtke convection scheme (Cristiano Padrin, CINECA)
  - Graupel microphysics (Bachelor student, ETH)
- Inter-GPU communication in Fortran code
- Integration and testing of GPU version

Demonstrator (HP2C OPCODE)

- Prototype implementation of the COSMO production suite of MeteoSwiss making aggressive use of GPU technology

144 CPUs
(1728 cores)

1 cabinet Cray XE6

8 CPUs (128 cores)

8 GPUs (19968 cores)
+ 2 CPUs (32 cores)

18cm

Tyan Server FT77A

- Same time-to-solution on substantially cheaper hardware:

  Factor ~3x in price, factor ~9x in power consumption

  Reduction in infrastructure costs

Milestones of HP2C OPCODE

2012

- **August** — **Dynamical core on single GPU**
- **November** — **Timeloop on single GPU**
  full timestep on single GPU, not optimized, real boundary conditions
- **December** — **Realcase on single GPU**
  verified and ready for real cases on single GPU

2013

- **April** — **Realcase on multiple GPUs**
  first real case simulations on multiple GPUs
- **June** — **COSMO Demonstrator running**

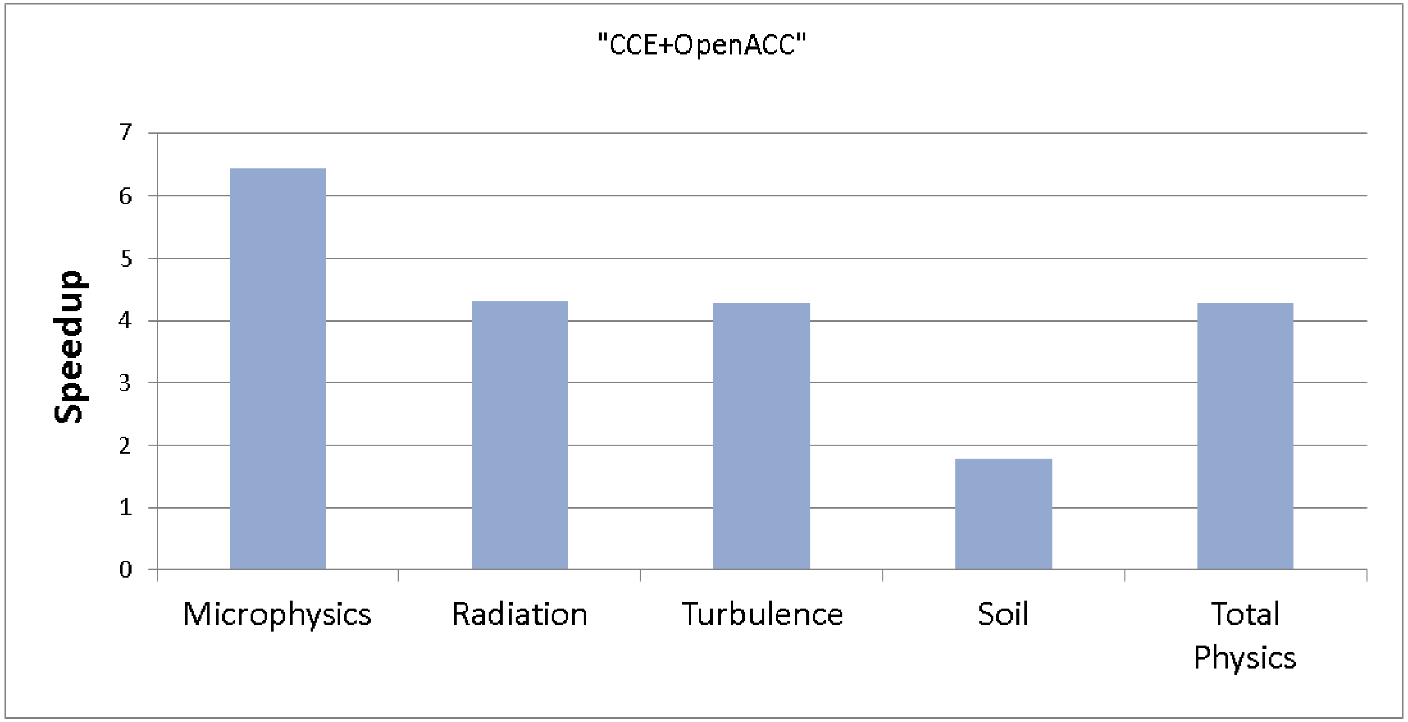
Conclusions

- **Complete rewrite of dynamical core using stencil library**
  - Single source code for GPU and CPU
  - Modern software engineering
  - Speedup of 2x for CPU and 5x for GPU
- **Porting of rest of code using compiler directives**
  - Physics (Speedup 4x for GPU)
  - Assimilation (no speedup)
- **Demonstrator by June 2013!**
- **Completion and integration of these developments into COSMO code until 2014**

Thank you for your attention

Performance of Physics

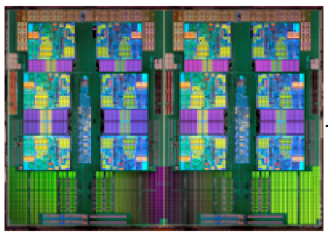
- Test domain 128x128x60 – 16 cores CPU vs GPU (Kepler)



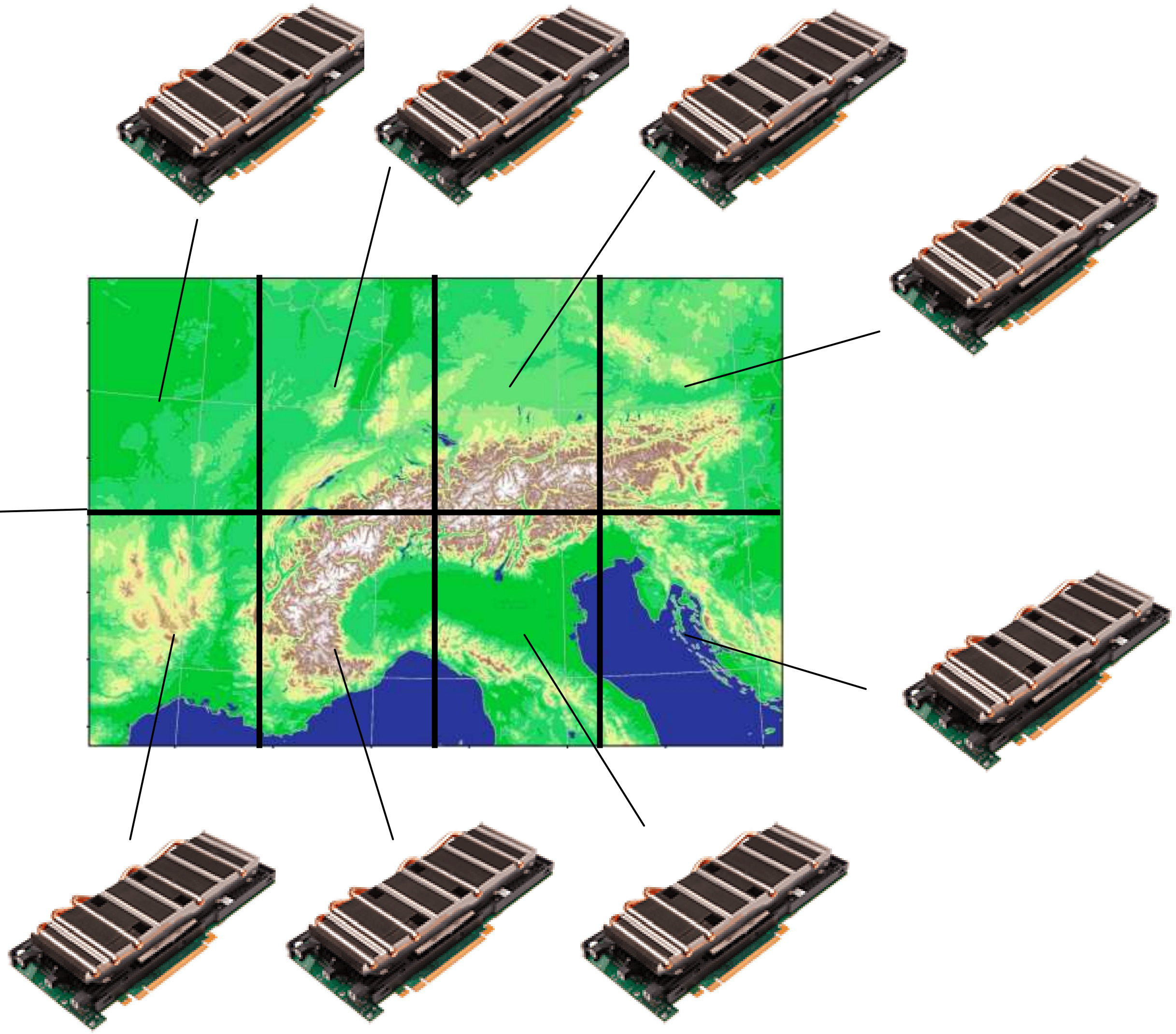
- Overall speed up ~4x
- Running the GPU-Optimized code on CPU is about 25% slower
  → separate source code for time critical routines

Running COSMO on a hybrid system

Domain decomposition
with 1 CPU core and
1 GPU per subdomain



Multicore CPU



Dycore: Sandy Bridge vs. Kepler