C2SM Center for Climate Systems Modeling

Porting the radiation parametrization on GPUs using directives

X. Lapillonne, O. Fuhrer

HP2C

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

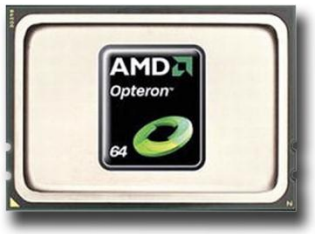
Eidgenössisches Departement des Innern EDI
Bundesamt für Meteorologie und Klimatologie MeteoSchweiz

Outline

- **Computing with GPUs**
- **Radiation scheme implementation**
- **Summary**

Computing on Graphical Processing Units (GPUs)

- Benefit from the highly parallel architecture of GPUs
- Higher peak performance at lower cost / power consumption.
- High memory bandwidth

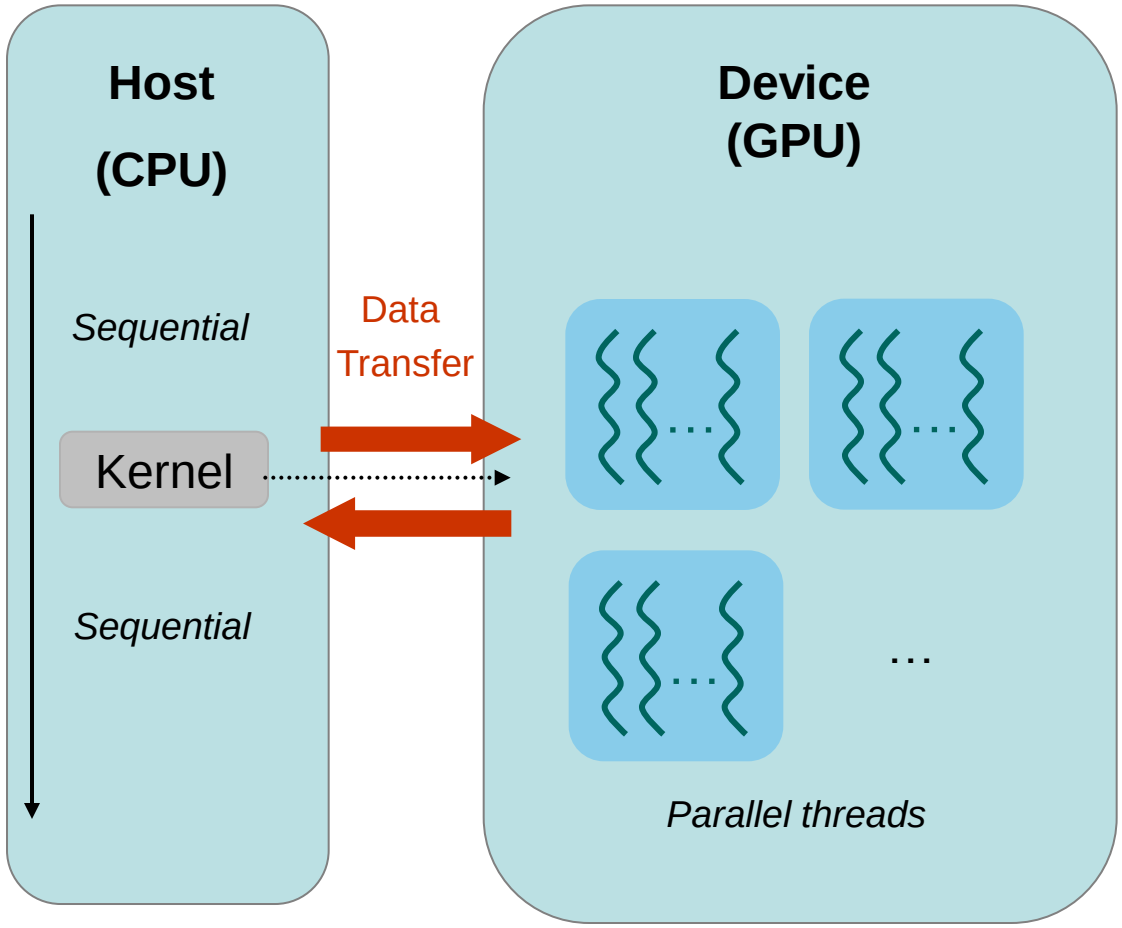


	Cores	Freq. (GHz)	Peak Perf. S.P. (GFLOPs)	Peak Perf. D.P. (GFLOPs)	Memory Bandwith (GB/sec)	Power Cons. (W)
CPU: AMD Magny-cours	12	2.1	202	101	42.7	115
GPU: Fermi M2050	448	1.15	1030	515	144	225

X 5 X 3

Execution model

- Copy data from CPU to GPU (CPU and GPU memory are separate)
- Load specific GPU program (Kernel)
- Execution: Same kernel is executed by all threads, SIMD parallelism (Single instruction, multiple data)
- Copy back data from GPU to CPU



Computing on Graphical Processing Units (GPUs)

- To be efficient the code needs to take advantage of fine grain parallelism so as to execute 1000s of threads in parallel.

- GPU code:
  - Programming level:
    - OpenCL, CUDA, CUDA Fortran (PGI) …
    - Best performance, but require complete rewrite
  - Directive based approach:
    - OpenMP-acc (CRAY), PGI, HMPP, F2ACC …
    - Smaller modifications to original code
    - The resulting code is still understandable by Fortran programmers and can be easily modified
    - Possible performance sacrifice with respect to CUDA code
    - New standard OpenACC (Cray, PGI, HMPP)

Key requirement for physical parametrizations

- Particular care concerning data management between host CPU and GPU is to be taken to get optimal performance :

  → In COSMO all computations shall be done on the GPU.

The directive approach, an example

note : PGI directives

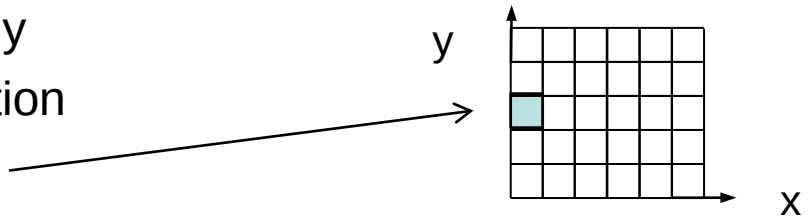
```
!initialization

do k=1,nlev
   do i=1,N
     a(i,k)=0.0D0
   end do
end do

! first layer

do i=1,N
  a(i,1)=0.1D0
end do

! vertical computation

do k=2,nlev
  do i=1,N
     a(i,k)=0.95D0*a(i,k-1)+exp(-2*a(i,k)**2)*b(i,k)
   end do
 end do
```

3 different kernels

Array “a” remains on the GPU between the different kernel calls

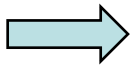
Implementation strategy for the parametrizations

- COSMO physics represents about 40 000 lines of codes: needs efficient implementation strategy
- Parallelization: horizontal direction
  1 thread per vertical column



- Most loop structure unchanged, one only adds directives
- In some parts, loop restructuring to reduce multiple kernel overhead call

```
!vertical loop
do k=2,nz
 !work 1
 do j=js,je
   do i=is,ie
     some work1(i,j,k,k-1)
    end do
 end do

 !work 2
 do j=js,je
  do i=is,ie
    some work2(i,j,k,k-1)
  end do
 end do

end do
```



```
!vertical loop
do k=2,nz
do j=js,je
   do i=is,ie
    !work 1
     some work1(i,j,k,k-1)

     !work 2
     some work2(i,j,k,k-1)
  end do
 end do
end do
```

- Remove Fortran automatic arrays in subroutines that are often called
- Data region to avoid CPU-GPU transfer
- Use profiler to target specific regions which need further optimization

Outline

- Computing with GPUs
- **Radiation scheme implementation**
- Summary

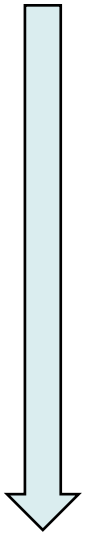
The radiation scheme

- Ritter-Geleyn scheme solves Radiative Transfer Equation (RTE)

$$\begin{aligned}
\frac{d\,F_u}{d\,\delta} &= \alpha_1\,F_u - \alpha_2\,F_d - \alpha_3 J \\
\frac{d\,F_d}{d\,\delta} &= \alpha_2\,F_d - \alpha_1\,F_d + \alpha_4 J \\
\frac{d\,S}{d\delta} &= -\alpha_5 S
\end{aligned}
\quad \Rightarrow \quad
\begin{bmatrix}
I & B_1 & & \\
A_2 & I & B_2 & \\
 & & \ddots & \\
 & & A_{n_z} & I
\end{bmatrix}
\begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_{n_z} \end{bmatrix}
=
\begin{bmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_{n_z} \end{bmatrix}$$

- $F_{u,d}$ vertical upward and downward fluxes, S parallel solar flux and δ is the optical depth; partial cloudiness is accounted for by assuming cloud free and cloudy contributions
- One such system is solved per spectral interval and gazes coefficient contribution
- Can be expressed in terms of a block tridiagonal linear system
  A_k, B_k are sparse 6x6 coefficient matrices
- The sytem is solved using an adapted elimination-backsubstitution algorithm.

Schematic of the radiation scheme



1. *precompute* : calculate some 3d coefficients, in particular cloud geometry

2. *loop ispec* : loop over thermal spectrum
   2.a *call opt th* : optical properties
   2.b *loop icoef* : loop over various absorption coefficients
   call inv th : solve linear RTE system
3. *loop ispec* : loop over solar spectrum
   3.a *call opt so* : compute optical properties and layers optical depth
   3.b *loop icoef* : loop over various absorption coefficients
   call inv so : solve linear RTE system

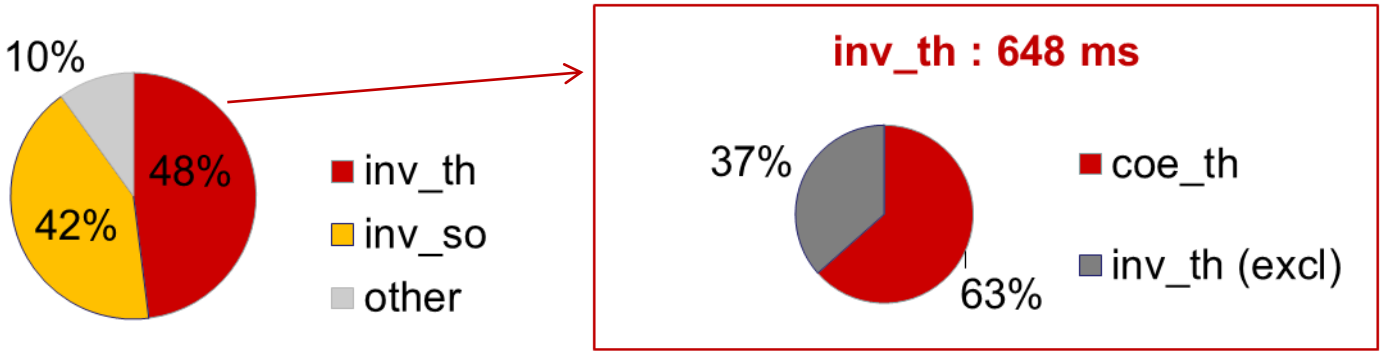
- The scheme is relatively costly, and call frequency to the radiation is adapted so that it represents about 8% of total execution time

CPU/GPU timing comparison

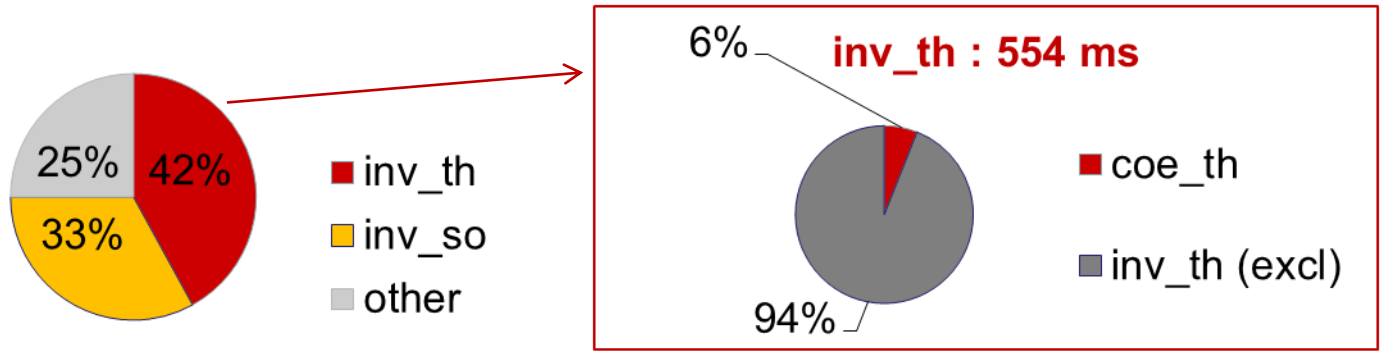
- Standalone code tested using domain of size n_x x n_y x n_z= 100x100x60
- CPU, 12 cores CPU : 1349 ms
- GPU, Fermi M2050 : 1108 ms (execution)
  32 ms (data transfer)

Profiling information

CPU :
1349 ms



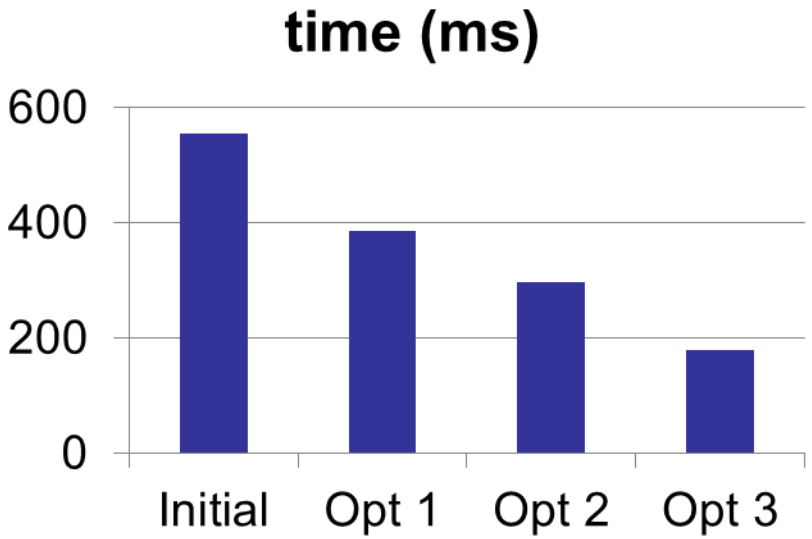
GPU :
1108 ms



- Most time spent in inv_th/so for both CPU and GPU
- Inside inv_th:
  - CPU: - most time spent in coe_th (compute bound, several exp)
    - linear system resolution, inv_th (excl), is computed in L2 cache
  - GPU: - linear system resolution performs poorly (memory bound)

GPU optimization for inv_th

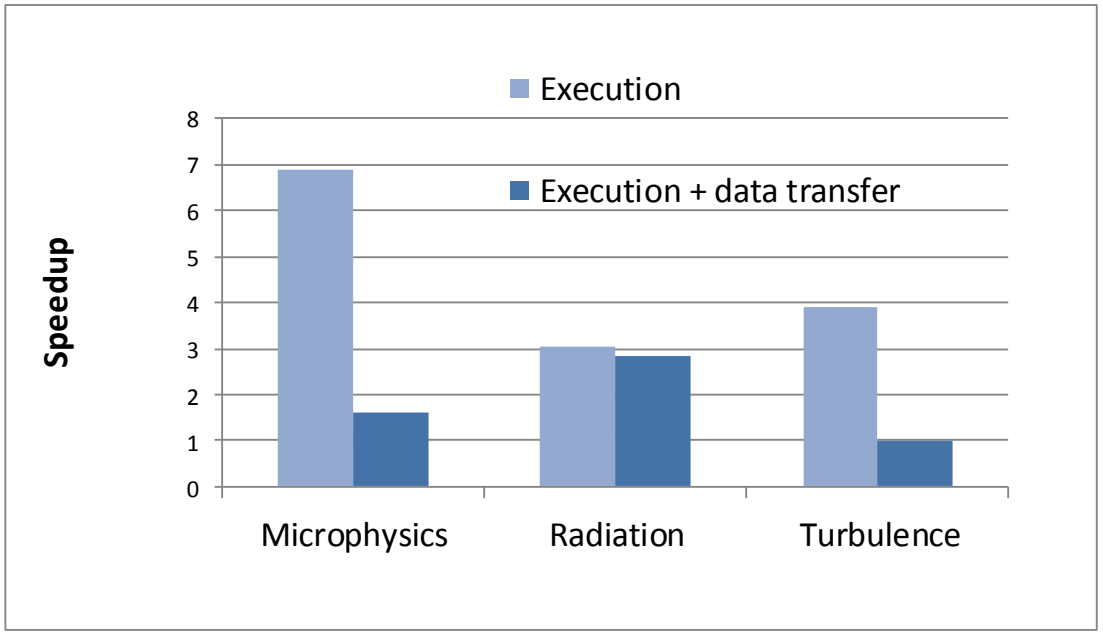
- Aim : reduce memory access
- Opt 1 : on the fly computation of cloud geometrical coefficients
- Opt 2: save array coefficients in scalars in the elimination step. Increase configurable cache size.



- Opt 3: Change data layout from t(nx,ny,nz) to t(nproma,nz) with nproma=nx x ny. This improves memory access pattern.
- The different optimizations lead to a factor x3 reduction in inv_th execution time
- With similar optimization in inv_so, the total execution time of the radiation scheme reduces to 442 ms, giving a final x3.1 speed up factor with respect to CPU.

Results for other parametrizations

- Applying similar implementation strategy the following speed up are obtained with respect to CPU code



- Overall speed up between x3 and x7
- For the microphysics and turbulence scheme the data transfer time is large with respect to execution time.

Outline

- Computing with GPUs
- Radiation scheme implementation
- **Summary**

Summary

- Compiler directives are well adapted for a GPU implementation of the physical parametrizations
- They allow to easily port large portion of code with minor modifications
- Profiler tools can then be used to identify part with low performance
- Compute intensive part map very well on the GPU (coe_th), memory intensive part may require special treatment
- Even if they do not provide full control on the GPU hardware, it is possible to optimize GPU code in this frame work. A factor x3 improvement was obtained for the inv_th routine in the radiation
- Comparing with CPU time, an overall speed up between x3 and x7 is obtained for the radiation, turbulence and microphysics parametrizations.
- These performances are within expectations and show that we are able to efficiently run on GPUs
- In general data transfer should be avoided and all computation should be done on the GPU. For the case of the radiation, this cost is however reduced with respect to execution time, and this scheme is a good candidate for hybrid computing.

Thank you

Additional slides

Influence of the number of horizontal points