Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

PaGUSci: Parallelization and Porting to GPU of scientific codes

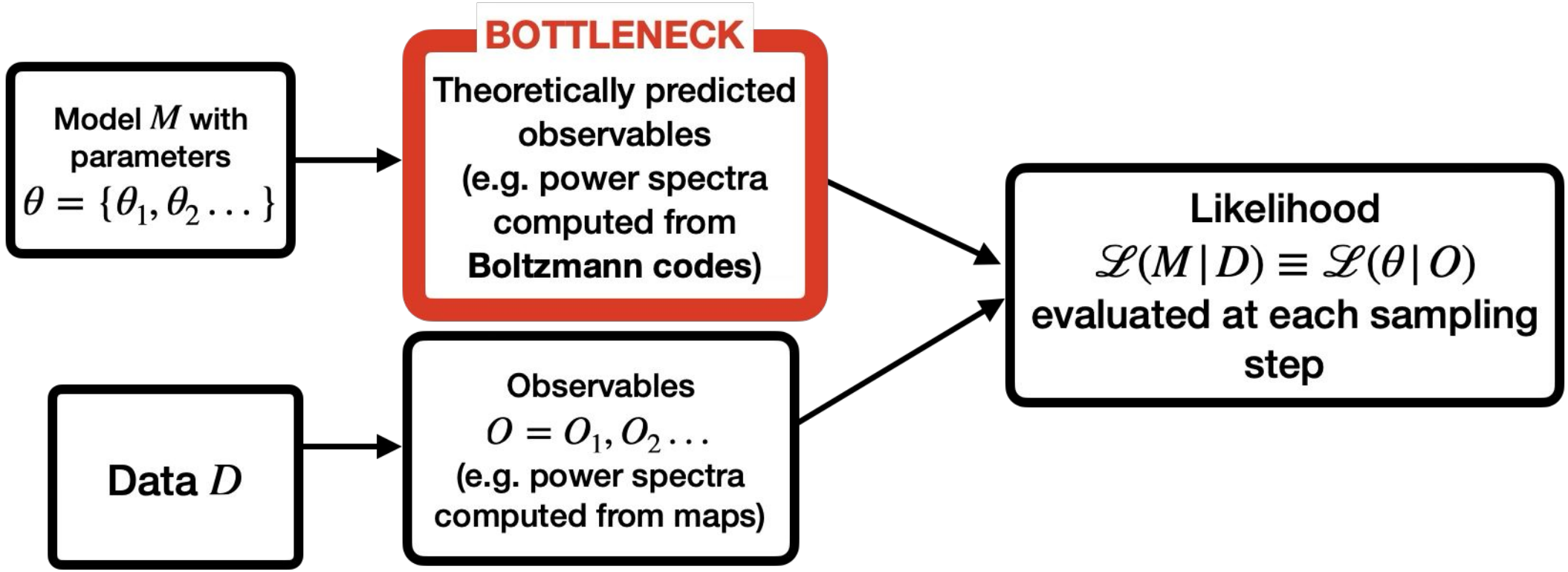
Loriano Storchi, Paolo Campeti, Nicolo' Antonini, Massimiliano Lattanzi

INAF ISTITUTO NAZIONALE DI ASTROFISICA NATIONAL INSTITUTE FOR ASTROPHYSICS
Istituto di Astrofisica Spaziale e Fisica cosmica di Milano

OAVDA

Spoke 3 III Technical Workshop, Perugia 28 Maggio, 2025

Scientific Rationale



- Solving cosmological Einstein-Boltzmann system of coupled ordinary differential equations (ODEs) for the cosmological perturbations
- Must be computed at each point in parameter space in typical MCMC
- **Carbon footprint**: Estimated $\mathcal{O}(100)$s of billions of CPU hrs/yr in the worldwide community

Scientific Rationale

- Bottleneck can be overridden by NN **emulators** (e.g. COSMOPOWER, Capse.jl)
- Emulators are $\mathcal{O}(10^{3-6})$ times faster than Boltzmann solvers! But…
- …require long **training** and **re-training** for each cosmological model being compared to data!
- Especially expensive for Beyond-ΛCDM models, e.g. including non-zero curvature or massive species and non-linear corrections
- Current/future surveys (ACT, Simons Observatory, CMB-S4, Euclid) require additional time for high accuracy spectra
- → **Training becomes very expensive!** $\mathcal{O}(10^{5-6})$ spectra needed/model, $\mathcal{O}(10)$ mins/spectrum
- Possible **Solution: porting Boltzmann codes CAMB or CLASS to GPU**

Technical Objectives, Methodologies and Solutions

- The two main **bottlenecks** in Einstein-Boltzmann Solvers are:

  1. The integration of cosmological perturbations over time that provides all transfer and source functions $S_P(k,\tau)$.

  2. The line-of-sight integrals which project the transfer function from Fourier space to harmonic space (with radial projection function ϵ related to Bessel functions)

$$C_\ell^{XY} = 4\pi \int \frac{dk}{k} \mathcal{P}_\mathcal{R}(k) \Delta_\ell^X(k) \Delta_\ell^Y(k)$$

$$\Delta_\ell^E(k) = \int d\tau S_P(k,\tau) \epsilon_\ell(k\,[\tau_0 - \tau]) \ .$$

- Step 1: perturbation module ideally suited for an NN approach (e.g. ClassNet), hard to optimize with HPC techniques: ODE algorithms are sequential in nature. Speedup factor ~ 3.

- Step 2: **harmonic transfer module** easier to tackle with **GPU porting**! Large number of independent integrals.

Technical Objectives, Methodologies and Solutions

Step 1: CAMB code profiling:

The profiling data show that __cambmain_MOD_doflatintegration is called 23,614 times, so that while the average time spent in this function per call is low, it is called frequently, which contributes to its high total time. This allowed us to identify the main loop responsible for most of the computational burden:

file cmbmain.f90:

```
272       do q_ix=1,ThisCT%q%npoints
273          call SourceToTransfers(ThisCT, q_ix)
```

Indeed the same loop has been already parallelized using OpenMP (via an OMP PARALLEL DO).
And Indeed an important portion of the total (up to 90% depending on the input) time is related to thi**s “Main_Task”.**

Technical Objectives, Methodologies and Solutions

Step 2: define a proper input

The profiling data show that the function __cambmain_MOD_doflatintegration takes up most of the time

But considering the total wall time taken (few seconds) we could not expect an intesrestin g beneefts coming from the GPU porting

We established three possible input scenario: **LOW, MEDIUM, HIGH**

CINECA Leonardo Serail code	Total Wall time (s.)	Main_Task Wall Time (s.)
LOW	27.28	1.41
MEDIUM	618.70	472.93
HIGH	6383.00	6158.81

Technical Objectives, Methodologies and Solutions

Step 3: compile the code using NVIDIA/PGI compiler

To initially simply compile the CAMB code we have been forced to quite a wide code refactoring/duplication because of many hacky and/or non fully standard code

call C_F_PROCPOINTER(c_funloc(fin), f) in MathUtils.f90

Unlike C, Fortran does not enforce left-to-right evaluation nor short circuiting

if (.not. allocated(ajl) .or. any(ubound(ajl) < [num_xx, max_ix])) then

Technical Objectives, Methodologies and Solutions

Step 4: Porting of the Main_Task to the GPU

Various issues , first instance clearly we need to well define all the data that is needed reason why:

1. Changed subroutines and functions API
2. Moved methods to standalone functions/subroutines
3. Minimize the code in the GPU
4. Because of something wrong in how the polymorphic types are getting set up in a few of the OpenMP regions

I filed an issue report, TPR #37404, tested your code with our next gen

```
.gitmodules                      |   3 +
camb/__init__.py                 |   2 +-
camb/camb.py                     |  30 +-
fortran/CPU/Makefile             | 179 ---------
fortran/GPU/Makefile             | 179 ---------
fortran/GPU/diffa.sh             |   5 -
fortran/Makefile                 | 238 +++++-------
fortran/Makefile_main            |   3 +-
fortran/MathUtils.f90            |   4 +-
fortran/bessels.f90              |   6 +-
fortran/callinganddta            |   3 -
fortran/camb.f90                 | 351 +-----------------
fortran/cmbmain.f90              | 722 ++++++++++++++++++++++++-----------
fortran/equations.f90            |  28 +-
fortran/omp/Makefile             | 179 ---------
fortran/openaccfunc.f90          | 686 ---------------------------------
fortran/openaccfunc_minimal.f90  | 426 ----------------------
fortran/params_high.ini          | 283 ---------------
fortran/params_low.ini           | 284 ---------------
fortran/params_medium.ini        | 284 ---------------
fortran/recfast.f90              |   6 +-
fortran/results.f90              | 776 +++++++++++++++++++++++++++++++++++++
fortran/run_comp.sh              |  43 +--
fortran/run_comp_omp.sh          |  39 +-
fortran/serial/Makefile          | 179 ---------
fortran/serial/diffa.sh          |   5 -
fortran/subroutines.f90          |  45 +--
forutils                         |   1 +
28 files changed, 554 insertions(+), 4435 deletions(-)
```

Main Results

- Inline with the timescale we have a first GPU running version of the CAMB code ported to GPU

Running CINECA Leonardo PRELIMINARY LOW

	Total Wall Time (s.)	Main_Task Time (s.)	Main_Task Speed up	Overall Speed up
OpenMP 32 threads (CPU)	1.61	0.09	15.45	16.95
OpenACC (GPU)	17.79	0.53	2.67	1.53
Expected OpenMP/OpenACC (CPU/GPU)	2.05	0.53	2.67	13.34

Maximum numerical difference with respect to the serial code 0.26%

Main Results

- Inline with the timescale we have a first GPU running version of the CAMB code ported to GPU

Running CINECA Leonardo PRELIMINARY MEDIUM

	Total Wall Time (s.)	Main_Task Time (s.)	Main_Task Speed up	Overall Speed up
OpenMP 32 threads (CPU)	33.13	26.96	17.57	18.67
OpenACC (GPU)	103.59	14.55	32.51	5.97
Expected OpenMP/OpenACC (CPU/GPU)	20.76	14.55	32.61	29.8

Maximum numerical difference with respect to the serial code 0.04%

Main Results

- Inline with the timescale we have a first GPU running version of the CAMB code ported to GPU

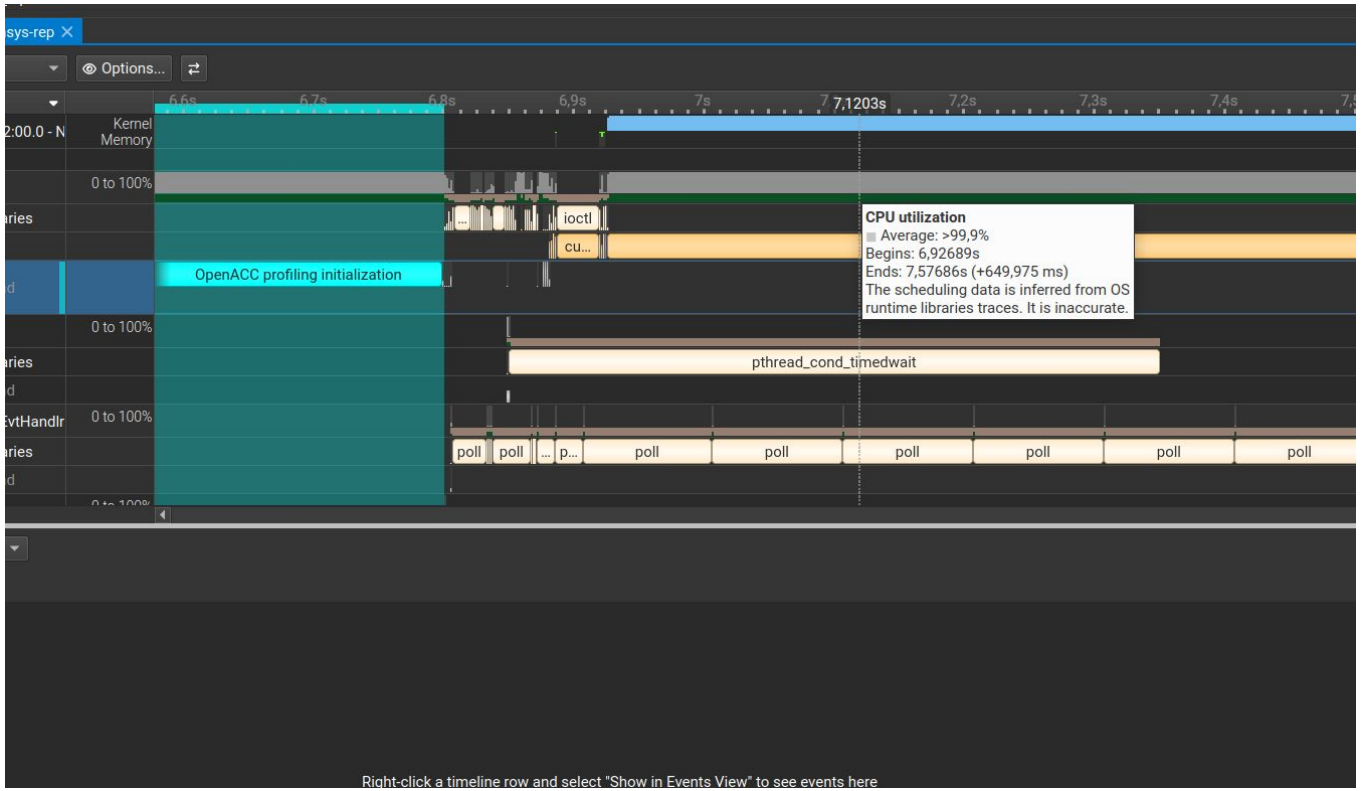
Running CINECA Leonardo PRELIMINARY HIGH

	Total Wall Time (s.)	Main_Task Time (s.)	Main_Task Speed up	Overall Speed up
OpenMP 32 threads (CPU)	348.38	329.67	18.68	18.32
OpenACC (GPU)	316.34	141.84	43.42	20.18
Expected OpenMP/OpenACC (CPU/GPU)	160.53	141.84	43.42	39.76

Maximum numerical difference with respect to the serial code 0.04%

Main Results

- Started profiling the GPU code tested on several platforms/GPUs



Final Steps

Try to find possible workarounds to have the OpenMP (multi CPU) part of the code running using the PGI/NVIDIA compiler (or other compilers)

Fine tuning of the OpenACC part (profiling and test already on going)

Porting also of the Non-Flat geometry

Building a training set to be used by NN models (mass production)