gCAMB: A GPU-accelerated Boltzmann solver for next-generation cosmological surveys

Loriano Storchi[b,a,*], Paolo Campeti[c,d,*], Massimiliano Lattanzi[c], Nicoló Antonini[b], Enrico Calore[c], Pasquale Lubrano[a]

[a] *INFN sezione di Perugia, Via Pascoli, Perugia, 06123, Italy*
[b] *Dipartimento di Farmacia, Università degli Studi G. D'Annunzio, Via dei Vestini, Chieti, 66100, Italy*
[c] *INFN Sezione di Ferrara, Via Saragat 1, Ferrara, 44122, Italy*
[d] *ICSC Centro Nazionale "High Performance Computing, Big Data and Quantum Computing", Via Magnanelli, Casalecchio di Reno, 40033, Italy*

Abstract

Inferring cosmological parameters from Cosmic Microwave Background (CMB) data requires repeated and computationally expensive calculations of theoretical angular power spectra using Boltzmann solvers like CAMB. This creates a significant bottleneck, particularly for non-standard cosmological models and the high-accuracy demands of future surveys. While emulators based on deep neural networks can accelerate this process by several orders of magnitude, they first require large, pre-computed training datasets, which are costly to generate and model-specific. To address this challenge, we introduce gCAMB, a version of the CAMB code ported to GPUs, which preserves all the features of the original CPU-only code. By offloading the most computationally intensive modules to the GPU, gCAMB significantly accelerates the generation of power spectra, saving massive computational time, halving the power consumption in high-accuracy settings and, among other purposes, facilitating the creation of extensive training sets needed for robust cosmological analyses. We make the gCAMB software available to the community.

Keywords: Numerical cosmology, Cosmic microwave background, GPU acceleration, High performance computing, Heterogeneous computing, Parallel algorithms, Energy-efficient computing

1. Introduction

Einstein-Boltzmann solvers (hereafter EBS) evolve cosmological linear perturbations for particle species (photons, baryons, cold dark matter, neutrinos, dark energy, etc.) coupled to the metric (Lifshitz, 1946; Lifshitz and Khalatnikov, 1963; Weinberg, 1972; Peebles, 1980; Ma and Bertschinger, 1995). EBS also deliver theoretical observables, such as CMB anisotropies and the matter power spectrum, which are necessary when fitting cosmological models to data. In modern cosmological inference, tens of millions of likelihood evaluations are routine to measure cosmological parameters, and the cumulative cost is dominated by computing these observables. EBS are therefore run massively by the worldwide cosmology community: it has been estimated that they are probably executed hundreds of billions of times every year (Albers et al., 2019), with substantial computational resource usage and associated electricity costs (and carbon footprint). The most commonly used EBSs are CAMB[1] (Lewis et al., 2000; Lewis and Challinor, 2011; Howlett et al., 2012) (in FORTRAN) and CLASS[2] (Lesgourgues, 2011) (in C), which exhibit comparable performances.

For minimal ΛCDM (i.e. with a cosmological constant and cold dark matter) Universes with standard settings, an EBS run can take well under a minute on a few CPU cores. Yet precise Bayesian inference, for instance, sweeps over tens of millions of models across multiple data combinations in order to provide posterior distribution for comsological parameters, turning this modest per-run time into a major bottleneck. The burden grows for extended (beyond-ΛCDM) scenarios, for instance including massive species like neutrinos (Bolliet et al., 2024; Jense et al., 2025) or nonzero curvature (Tian et al., 2021; Anselmi et al., 2023b,a). Moreover, because of high-accuracy requirements – particularly at small angular scales– required by future surveys, single evaluations can reach the $\sim$ 10-minute regime (Bolliet et al., 2024; Jense et al., 2025).

Contemporary neural network-based power spectrum emulators (such as COSMOPOWER (Spurio Mancini et al., 2022) and Capse.jl (Bonici et al., 2023)) aim to bypass repeated EBS calls, offering speedups of $\sim 10^{3-6}$. However, these gains shift the cost upstream: training typically requires $\sim 10^{5-6}$ spectra and retraining for each model variant, with the largest computational expense in non-standard cosmologies. Current (e.g. ACT (Henderson et al., 2016)) and upcoming (Simons Observatory (Simons Observatory, 2019) and Euclid (Laureijs et al., 2011)) surveys further tighten the accuracy and therefore computational demands (Bolliet et al., 2024; Jense et al., 2025).

In this work, we target this dual challenge—fast inference and affordable creation of training datasets for emulators—by offloading the most expensive CAMB modules to GPUs and making the resulting software implementation gCAMB publicly available (Storchi, a,b). Beyond speed, this approach leverages comparatively underutilized GPU resources and might reduce energy consumption in certain cases, addressing also sustainabil-

*Author to whom correspondence should be addressed
Email addresses: loriano@storchi.org, loriano.storchi@unich.it, loriano.storchi@pg.infn.it (Loriano Storchi), paolo.campeti@fe.infn.it (Paolo Campeti)

[1] https://github.com/cmbant/CAMB
[2] https://github.com/lesgourg/class_public

ity concerns alongside throughput. Notably, gCAMB preserves all the features of the CAMB code, so its adoption requires minimal changes to existing workflows. Moreover, since only the line-of-sight integration is offloaded to the GPU, the primordial and source-function modules remain essentially unchanged by the porting and can be modified as usual for testing specific theoretical models.

The paper is organized as follows. Section 2 reviews the internals of EBS, identifies GPU-amenable kernels, and outlines the scientific applications of gCAMB. Section 3 details the algorithms and implementation choices. Section 4 presents profiling and validation. Section 5 summarizes findings and future directions.

2. Einstein-Boltzmann solvers

2.1. Key equations

We briefly summarize here the inner working of the EBS, with some key equations. We refer to Refs. (Ma and Bertschinger, 1995; Seljak and Zaldarriaga, 1996) for a complete treatment in the context of cosmological perturbation theory.

After linearizing the Einstein–Boltzmann system, Fourier perturbation modes decouple and the equations for each wavenumber k can be solved independently via an ODE system in conformal time τ. This yields transfer functions $\Delta_\ell^X(k)$; these quantities encode the relation between the anisotropies of the measurable quantity X as observed from our position in space and time, and the primordial perturbations generated in the early Universe. The transfer functions are then used by the EBS to compute angular power spectra C_ℓ^{XY} from

$$C_\ell^{XY} = 4\pi \int \frac{dk}{k}\, \mathcal{P}_\mathcal{R}(k)\, \Delta_\ell^X(k)\, \Delta_\ell^Y(k)\,, \tag{1}$$

with $X, Y = \{T, E\}$, and $\mathcal{P}_\mathcal{R}(k)$ is the power spectrum of primordial perturbations. The angular power spectra are the observed quantities that can be predicted by the theory and on which cosmological likelihoods are tipically based. They describe, in harmonic space, the angular correlation between observables X and Y. In the following, we will concentrate on CMB temperature (T) and E-mode polarization (E) anisotropies.

EBS compute the transfer functions in the so-called *line-of-sight* formalism, from gauge-invariant CMB source functions for temperature $S_T(k,\tau)$ and polarization $S_P(k,\tau)$ fluctuations. Assuming scalar-only perturbations sourcing only T and E modes (similar equations arise in the case of tensor perturbations and B modes), these source functions are projected to harmonic space as (Seljak and Zaldarriaga, 1996)

$$\Delta_\ell^T(k) = \int d\tau\, S_T(k,\tau)\, \varphi_\ell[k(\tau_0-\tau)], \tag{2}$$

$$\Delta_\ell^E(k) = \int d\tau\, S_P(k,\tau)\, \epsilon_\ell[k(\tau_0-\tau)]\,. \tag{3}$$

with $\varphi_\ell, \epsilon_\ell$ kernels related to spherical Bessel functions in a spatially flat Universe (and to hyperspherical Bessel functions in a non-flat one).

2.2. Computational bottlenecks

The two main bottlenecks common to EBSs (both in CLASS and CAMB) are (Albers et al., 2019):

1. The *source functions calculation*, i.e. the integration of the k-wise ODEs of cosmological perturbations over time τ to obtain source functions in $S_T(k,\tau)$ and $S_P(k,\tau)$;
2. The *line-of-sight integration*, i.e. the highly oscillatory, massively repeated projections $(k,\tau) \to (\ell, k)$ in Eqs. 2-3 above.

Which step dominates depends on context: computing CMB angular power spectra spectra for ΛCDM cosmologies up to small scales (i.e. to high multipoles), the projection stage in step 2 often wins; for extended physics (massive/interactive species, modified gravity) even basic outputs make step 1 dominant; in many ΛCDM use-cases they are comparable.

From a computational standpoint, the source function calculation in step 1 parallelizes only over the outer k-loop; its sequential time-stepping limits strong scaling (saturating around $\sim$ 10 threads), making it a poor target for brute-force parallelization over many CPU/GPU cores. This is why neural network-based emulation of source functions—e.g., the COSMICNET approach (Albers et al., 2019; Günther et al., 2022)—has proven effective, yielding a factor ~ 2–3 of end-to-end speedup for spectra while preserving accuracy. In contrast, the harmonic-transfer stage consists of vast numbers of independent integrals which is an embarrassingly parallel workload. Further algorithmic reformulations could also reduce the cost of oscillatory integrals (e.g. (Assassi et al., 2017; Schöneberg et al., 2018)).

In what follows we focus on the CAMB code, targeting specifically the line-of-sight integration step with GPU porting and exploiting fine-grained parallelism at scale to accelerate it.

3. Computational details

In the present section, we will describe all the computational details related to the porting of the CAMB code to the GPU. We start from a brief description of the parameter settings we used to benchmark gCAMB (Storchi, a) against CAMB (Lewis and Challinor).

3.1. Cosmological and accuracy parameters for benchmarking

To assess performance as a function of numerical accuracy, we benchmark gCAMB against CAMB under three accuracy settings – which we refer to as *Low*, *Medium* and *High* in the following – while holding the cosmological parameters fixed to *Planck* measured values (Aghanim et al., 2020). The three configurations for benchmarking are summarized in Table 1, where also the corresponding wall clock times on 32 CPU cores are reported.

In all runs we compute both scalar and tensor spectra (`get_scalar_cls=True`, `get_tensor_cls=True`). For scalars we set `l_max_scalar=11000` (maximum multipole for

Table 1: Accuracy parameter configurations used for benchmarking and corresponding wall times for the CAMB code execution measured on 32 CPU cores on an Intel Xeon Platinum (PI) 8358 CPU with 2.6 GHz. See Section 3 for details.

Setting	accuracy_boost	l_sample_boost	Wall time (32 cores)
Low	1	1	1.56 s
Medium	3	20	32.18 s
High	3	50	329.92 s

scalar C_ℓ[3]) and k_eta_max_scalar=22000 (which effectively sets the wavenumber k range used in line-of-sight projections for scalar perturbations); for tensors l_max_tensor=1500 (maximum tensor multipole) and k_eta_max_tensor=3000 (wavenumber cutoff for tensor similarly to scalars). The three accuracy configurations use accuracy_boost (i.e the global step-size/tolerance scaler[4]) and l_sample_boost (the output ℓ-sampling density, i.e., number of explicitly computed multipoles[5]) as reported in Table 1. All other CAMB accuracy parameters are kept to their default value in the params.ini parameter file shipped with the code. We refer to the extensive CAMB documentation[6] – specifically we refer to the FORTRAN version – for detailed explanation of all the accuracy parameters mentioned above.

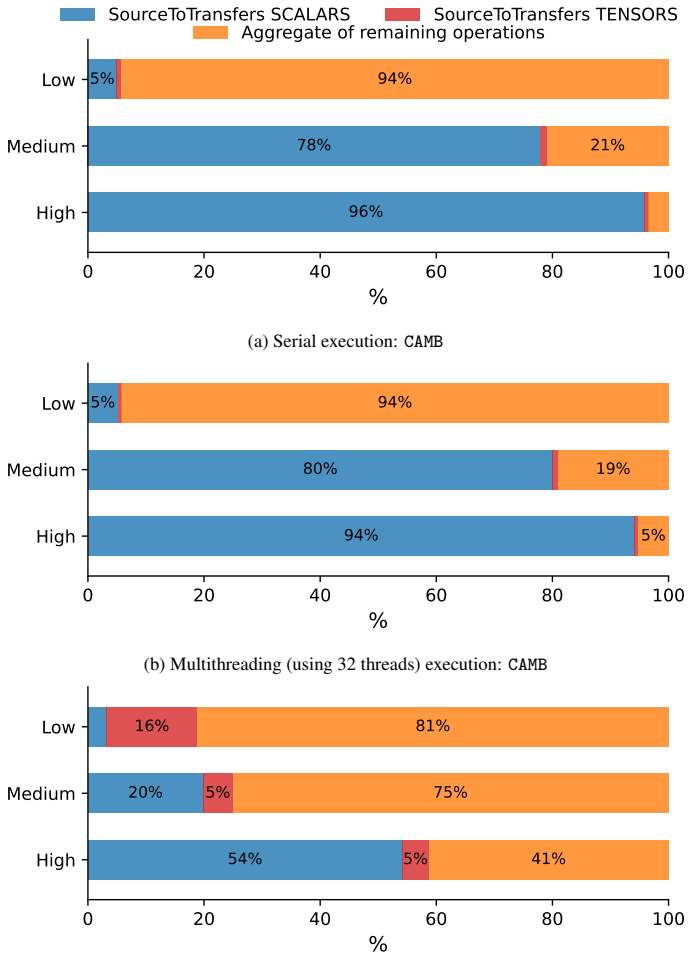
We increased l_sample_boost to load the GPU-accelerated line-of-sight integrator—while leaving l_accuracy_boost untouched because it would only slow down hierarchy evolution, which we do not attempt porting to GPU; this stresses the part we optimized and prepares for cases that truly need ℓ-by-ℓ precision (such as primordial features, curved/anisotropic cosmologies). We note though that standard ΛCDM emulators can typically get by with the default ~ 1.5 sampling.

While for our benchmarks we scale accuracy also via the accuracy_boost parameter (following Bolliet et al. (2024)), we recommend that for production runs one identifies the dominant numerical error and tune the specific sub-parameters responsible—rather than globally increasing accuracy_boost — as this is typically faster and can yield higher accuracy than the uniform scaling it implies.

Regarding the need for high-accuracy settings, we note also that in practice, pushing C_ℓ to accuracy far beyond what affects post-marginalization cosmology is wasted effort and for $\ell \gg 10^3$ the dominant uncertainties are from the nonlinear lensing model—so experiment-realistic error budgets, not maximal numerical precision, should set the target.

3.2. GPU porting of the code: gCAMB

As discussed in Section 2.2, we expect one of the two main bottlenecks in CAMB execution to be the line-of-sight integration of the CMB source functions, a task which we will name from now on SourceToTransfers. This is confirmed by profiling CAMB on CPUs (panel (*a*) and (*b*) of Figure 1). Details on how we profile CAMB are given in Section 4.



(a) Serial execution: CAMB

(b) Multithreading (using 32 threads) execution: CAMB

(c) Hybrid GPU (NVIDIA A100) and multithreading (using 32 threads) execution: gCAMB

Figure 1: Percentage of wall-time taken by the two SourceToTransfers tasks (one for scalar perturbations and the other for tensor ones), with the remaining operations involved in the complete run reported as "Aggregate of remaining operations". Panel (*a*) shows the results for the serial execution of CAMB, while panel (*b*) shows CAMB executed on 32 OpenMP threads. Finally, panel (*c*) shows the hybrid GPU+CPU execution of gCAMB on 32 threads.

To adapt the CAMB code for GPU we used the OpenACC framework, a directive-based programming allowing developers to offload parts of their code to accelerators like GPUsWienke et al. (2012). We began by extracting some of the SourceToTransfers task related methods from their main modules such as CAMBmain. We refactored these methods into standalone functions, which allowed us to precisely identify and minimize the input and output data that must be moved between the CPU and the GPU. This strategy proved to be highly efficient, as it allowed us to create a minimal code package for the GPU, avoiding the transfer of larger, unnecessary code sections.

The final product is a streamlined code tailored specifically for the flat universe scenario. The operational flow of this ported code is summarized in the pseudo-code reported in Al-

[3]Near $\ell_{\rm max}$ the accuracy of the spectra degrades; for lensed spectra one typically need some headroom above the target ℓ range, as recommended in the CAMB documentation.

[4]Higher values of accuracy_boost tighten time steps, tolerances, and k-sampling across all modules (except the ones controlled by l_sample_boost and l_accuracy_boost).

[5]Higher values reduce interpolation error; for l_sample_boost=50 CAMB computes all ℓ explicitly).

[6]https://camb.info/

gorithm 1, where we outline the primary `SourceToTransfers` task for the accelerator not encompassing the complete calling sequence or all operations, but rather aiming to provide a concise overview of the ported procedure. Indeed, the overall computation is handled by the loop over the number of points that is implemented in the `TimeSourcesToCl` subroutine within the `CAMBmain` module. Finally, it is the main routine `SourceToTransfers`, within the main loop, that orchestrates the entire process. This routine begins by calling `IntegrationVars_init` to set up the necessary integration variables and `InterpolateSources` to prepare the source data. The core of the calculation is performed by the `DoSourceIntegration` function. Within this function, the majority of the computational effort is spent in the `DoFlatIntegration` subroutine. The whole procedure can be summarized as follow:

- Data Transfer to GPU: The process starts by allocating memory on the GPU and copying the necessary input data from the main system's memory (CPU) to the GPU's memory. This is represented by the "begin accelerator data region" block in the pseudo-code 1.
- Coarse-Grained Parallelism (Gang): The outer loop runs over the index $q_{ix} = 1, \ldots, N$, each corresponding to a cosmological perturbation mode. In flat geometry, this is the comoving wavenumber k. This loop is parallelized at a high level (gang). Each iteration, which involves a call to the `SourceToTransfers` routine, is treated as an independent task assigned to a group of processing units on the GPU.
- Fine-Grained Parallelism (Vector): Inside the `SourceToTransfers` routine, there are several loops, of which the most demanding ones are the two nested inner loops reported in the pseudo-code. The first one is a vectorized loop over $j = 1, \ldots, L_{\max}$ corresponding to cycling over the multipoles ℓ. For each ℓ and source X (e.g., temperature, E/B polarization, lensing), the routine builds the transfer functions $\Delta_\ell^X(q)$. Nested within the ℓ-loop is a vectorized loop over $n = n_{\min}, \ldots, n_{\max}$ that cycles through discretized line-of-sight sampling points (conformal times η_n or equivalently comoving distances χ_n). This loop accumulates the line-of-sight integral in Eqs. 2, which involves calculation of the spherical Bessel-function. All of these loops are parallelized at a much finer level (vector). This means that the iterations within these loops are executed simultaneously by individual processing threads within a gang, which is highly efficient for array-based calculations such as the Bessel integration mentioned above.
- Data Transfer from GPU: Once all the parallel computations are finished, the resulting transfer functions—the 3D array $\{\Delta_\ell^X(q)\}$ over sources X, multipoles ℓ, and modes q—are copied from the GPU's memory back to the CPU's memory, and the memory on the GPU is freed up. This is marked by the "end accelerator data region" command in the pseudo-code reported in Algorithm 1.

Algorithm 1 The following pseudocode outlines the primary `SourceToTransfers` task ported to the GPU. This representation does not encompass the complete calling sequence or all operations, but rather aims to provide a concise overview of the ported procedure. See the text for a more detailed description of the full calling sequence.

```
1:  begin accelerator data region                    ▷ Copy data to GPU
2:  for each q_ix from 1 to N do in parallel (gang)
3:      SourceToTransfers(q_ix, ...)
            ▷ Inside SourceToTransfers (executed on GPU)
4:      for each j from 1 to L_max do in parallel (vector)
5:          ...                                      ▷ Initialize local sums
6:          for each n from n_min to n_max do in parallel (vector)
7:              ▷ Perform calculations (e.g., Bessel integration)
8:              ...
9:          end for
10:     end for
11: end for
12: end accelerator data region                      ▷ Copy results from GPU
```

4. Results and discussion

This section provides a detailed analysis of the numerical and timing results obtained from the `gCAMB` code. Our investigation focuses on evaluating the efficiency and accuracy of the `gCAMB` implementation, particularly in comparison to other existing versions or methodologies.

4.1. Computational environment and build configuration

All tests were performed with the Tier-0 EuroHPC supercomputer Leonardo at CINECA (Turisini et al., 2023) running an Intel Xeon Platinum (PI) 8358 CPU with 2.6 GHz (total 32 cores), in the case of the accelerated CPU/GPU version we use a single NVIDIA A100 GPU. When compiling the `gCAMB` code, we use the NVIDIA HPC compiler (nvfortran version 24.5-1) for CPU and CPU/GPU. Instead, we used the GNU Fortran compiler, version 8.5.0, when compiling the serial and the OpenMP (CPU only) versions of the code. This is due to a known issue of the NVIDIA `FORTRAN` compiler, detailed in Section 4.3.

4.2. Benchmark configurations and measured quantities

Table 2 provides a comprehensive overview of the performance characteristics of various code versions, focusing on the total execution time (walltime) and the specific time consumed by the crucial `SourceToTransfers` task. Given that both scalar and tensorial modes are processed, the `SourceToTransfers` procedure is invoked twice; consequently, both the individual timings for each execution and their cumulative duration are reported. The versions considered in this analysis include: "Serial (CPU)", "OpenMP (32 thr)", "`gCAMB` (GPU+CPU, 32 thr)" and "`gCAMB` (GPU+CPU, 32 thr, est.)". The "Serial (CPU)" represents the baseline, standard implementation of the `CAMB` code, executed sequentially without parallelization. It serves as a fundamental point

of comparison for evaluating the benefits of parallel computing. the "OpenMP (32 thr)" version leverages the Open Multi-Processing (OpenMP) application programming interface for parallel programming, specifically targeting CPU-only execution. We utilize 32 threads to distribute computational tasks across multiple cores. The "gCAMB (GPU+CPU, 32 thr)" represents a hybrid approach, integrating both GPU and CPU resources for computation. Like the OpenMP version, it also employs 32 threads, demonstrating an effort to accelerate performance by offloading the `SourceToTransfers` task to the GPU while retaining CPU involvement for other tasks.

4.3. Estimated fully OpenMP-enabled timing

Additionally, Table 2 includes a specific entry, "gCAMB (GPU+CPU, 32 thr, est.)", which presents an estimated walltime for a hypothetical, fully OpenMP-ported version of gCAMB. This estimation is a necessary inclusion due to current technical limitations encountered with the NVIDIA `FORTRAN` compiler. Specifically, the compiler exhibits restrictions regarding the proper setup of polymorphic types within certain OpenMP regions. These limitations necessitated the temporary removal of some OpenMP sections during the gCAMB compilation process, preventing a direct, fully OpenMP-enabled measurement. The estimated value thus provides valuable insight into the potential additional performance gains achievable once these compiler constraints are overcome and a complete OpenMP integration is feasible.

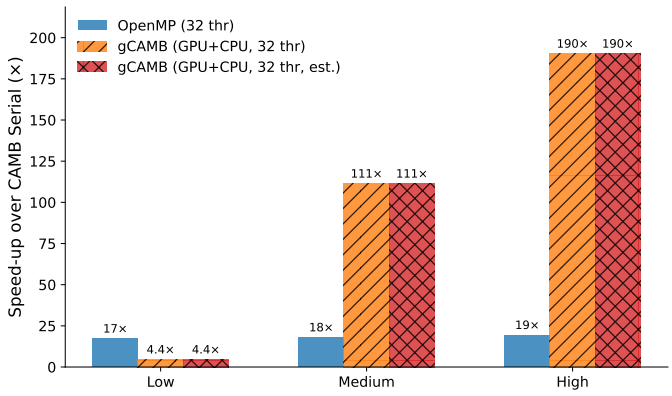
Estimation is achieved by first calculating the total walltime recorded for the OpenMP execution. From this total walltime, the duration attributed to `SourceToTransfers` operations is subtracted. Subsequently, the `SourceToTransfers` computation time derived specifically from the gCAMB code is instead added. This method aims to crudely estimate the walltime for a hypothetical, fully OpenMP-ported version of gCAMB. In general, the NVIDIA compilation issue is projected to affect less than 10% of the high- and low-accuracy scenarios. However, its influence is more significant in the medium accuracy scenario, leading to a performance reduction of up to 38%.

4.4. Performance results and speedups

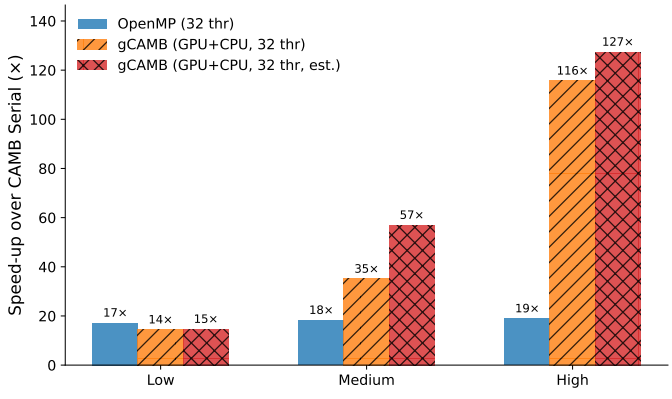
Based on the timing data in Table 2 and the speed-ups shown in Figure 2, the primary advantage of using gCAMB is the dramatic acceleration of the `SourceToTransfers` task, which is the main computational bottleneck in the standard CAMB code. This targeted optimization leads to a very significant reduction in the overall time required to generate cosmological power spectra, especially for the high parameters configuration. Indeed, as shown in Figure 2, the speed-up for the `SourceToTransfer` task alone is enormous. For the high accuracy setting, gCAMB achieves a speed-up of nearly 10x compared to the multi-threaded CPU version (OpenMP 32 Threads). Because `SourceToTransfers` dominates the total runtime (especially in medium and high accuracy scenarios, see also Figure 1), accelerating it provides a substantial overall performance boost. Figure 2 shows that for the high accuracy setting, the overall speed-up is over 100x. This transforms a calculation that takes over 5 minutes on 32 CPU cores (329.92s) into one that completes in under a minute (54.01s) on the GPU.

4.5. GPU warm-up behavior and the low-accuracy regime

It is clear that the advantage of gCAMB becomes more pronounced as the computational demand increases. Indeed, in the low accuracy scenario, the overhead of GPU warm-up time makes the benefit less obvious. It is a common and expected phenomenon for the first execution of a GPU-accelerated kernel to be significantly slower than subsequent runs, even when performing the exact same task. This initial delay is often referred to as a "warm-up" period and is caused by a combination of one-time setup and initialization overheads. Reason why the speed-up of the first `SourceToTransfers` task is generally always the worst respect to the second `SourceToTransfers` task execution. Clearly while in the medium and high accuracy scenarios the improvement in terms of speed-up in the tensors GPU execution is able to fully recover this warm-up time, this is not the case for the low accuracy scenario.



(a) `SourceToTransfers` Speed-up



(b) Overall Speed-up

Figure 2: Speed-up factors relative to the serial CAMB execution on a single CPU core for both the `SourceToTransfers` task (upper panel) and the end-to-end execution ("Overall"). We show the speed-up factors for the low, medium and high accuracy settings and for CAMB parallel execution on 32 CPU cores ("OpenMP 32 Threads"), hybrid GPU and CPU execution of gCAMB ("gCAMB 32 Threads)") and the estimated times for hybrid execution of gCAMB expected for the full OpenMP port (i.e. "gCAMB 32 Threads (Est.)").

Table 2: Wall-clock times (seconds) for CAMB/gCAMB under the three accuracy settings (low, medium and high) described in Section 3. For each setting, we report the execution times of CAMB on a single CPU core ("Serial (CPU)") and on 32 OpenMP CPU threads ("OpenMP (32 thr)"), hybrid GPU and CPU execution of gCAMB ("gCAMB (GPU+CPU, 32 thr)") and the estimated times for hybrid execution of gCAMB expected for the full OpenMP port (i.e. "gCAMB (GPU+CPU, 32 thr, est.)"). For each execution method, we report the runtimes of the SourceToTransfers routine for scalars and tensors separately and together ("Subtotal"). Finally, the last column ("All") shows the runtimes for end-to-end execution for each method. For further details see Section 4. All runs were performed on the Leonardo supercomputer at CINECA.

Accuracy	Run method	SourceToTransfers [s]			All [s]
		Scalars	Tensors	Subtotal	Total
Low	Serial (CPU)	1.32	0.22	1.54	26.79
	OpenMP (32 thr)	0.08	0.01	0.09	1.56
	gCAMB (GPU+CPU, 32 thr)	0.06	0.29	0.35	1.86
	gCAMB (GPU+CPU, 32 thr, est.)	0.06	0.29	0.35	1.83
Medium	Serial (CPU)	457.98	6.94	464.92	588.19
	OpenMP (32 thr)	25.76	0.28	26.04	32.18
	gCAMB (GPU+CPU, 32 thr)	3.33	0.85	4.18	16.78
	gCAMB (GPU+CPU, 32 thr, est.)	3.33	0.85	4.18	10.33
High	Serial (CPU)	5 983.50	43.92	6 027.42	6 246.63
	OpenMP (32 thr)	310.67	1.92	312.59	329.92
	gCAMB (GPU+CPU, 32 thr)	29.24	2.46	31.70	54.01
	gCAMB (GPU+CPU, 32 thr, est.)	29.24	2.46	31.70	49.04

4.6. Runtime composition and remaining acceleration opportunities

It should be noted that the computational load of SourceToTransfers constitutes, excluding the case of the low-accuracy scenario, a significant portion of the overall calculation, see Figure 1. Furthermore, it is evident that this percentage shifts considerably when the primary computational load is offloaded to the GPU, with the exception of the low accuracy scenario, which is influenced by the aforementioned "warm-up" period. Overall, this demonstrates that there is potential for further improvement by porting additional sections of the code to the GPU.

4.7. Reduction in power consumption

Before we start reporting the numerical results, it is also notable that we conducted a preliminary estimation of the power consumption of the CAMB code compared to the gCAMB one. Software measurements were performed on a machine equipped with an NVIDIA RTX 6000 Ada Generation and a dual AMD EPYC 9224 24-Core Processor, so that we could run performance analysis tools (as detailed below) with root permissions, which is not possible on the Leonardo computer cluster. Using 32 threads, for both CAMB and gCAMB execution, a significant decrease in power consumption was observed in the high accuracy scenario. In this scenario, the CPU-only code (CAMB) consumed 61283.24 Joules, while the gCAMB code, accounting for both CPU and GPU consumption, consumed 33042.42 Joules, representing an approximate halving of power consumption. All measurements were conducted using the perf tool (de Melo, 2010) for CPU energy consumption and nvidia-smi one (NVIDIA Corporation, 2025b) for GPU consumption (the energy consumption metric was not available for this GPU via the NVIDIA Nsight Compute tool (NVIDIA Corporation, 2025a)).

4.8. Numerical validation of power spectra

We quantify the numerical agreement between the spectra obtain from CAMB running on 32 OpenMP CPU threads and gCAMB hybrid GPU+CPU execution 32 OpenMP threads by the fractional difference per multipole,

$$\frac{\Delta C_\ell^X}{C_\ell^X} \equiv \frac{C_\ell^{X,\mathrm{GPU}} - C_\ell^{X,\mathrm{CPU}}}{C_\ell^{X,\mathrm{CPU}}}, \quad X \in \{\mathrm{TT}, \mathrm{TE}, \mathrm{EE}, \mathrm{BB}\}.$$

The fractional differences (one per spectrum, considering the usual low, medium and high accuracy settings) are shown in Fig. 3.

Additionally, we compare the difference between GPU- and CPU-computed spectra to the cosmic variance (assuming an ideal noiseless full-sky observation) $\sigma_{\mathrm{CV}}^{XY}(\ell)$ at each multipole ℓ. If $|\Delta C_\ell^{XY}| \ll \sigma_{\mathrm{CV}}^{XY}(\ell)$ we considered this test passed for gCAMB. The cosmic variance for auto (TT, EE, BB) and cross (TE) spectra[7] is given by

$$\sigma_{\mathrm{CV}}^{TT,EE,BB}(\ell) = \sqrt{\frac{2}{2\ell+1}}\left|C_\ell^{TT,EE,BB}\right|, \tag{4}$$

$$\sigma_{\mathrm{CV}}^{TE}(\ell) = \sqrt{\frac{(C_\ell^{TE})^2 + C_\ell^{TT} C_\ell^{EE}}{2\ell+1}}. \tag{5}$$

[7] We do not consider TB and EB here because they are expected to vanish in standard scenarios.

The $|\Delta C_\ell|/\sigma_{\rm CV}(\ell)$ for all spectra and the different accuracy settings is shown in Fig. 4, with the horizontal line at 1 marking the per–multipole cosmic-variance threshold.

Across all spectra and all multipoles we find $|\Delta C_\ell|/\sigma_{\rm CV}(\ell) < 1$. The worst case occurs at the highest multipoles in EE and BB, where the numerical differences still remain at least a factor of ~ 5 below full-sky cosmic variance; in all other regimes the differences are several orders of magnitude smaller. Although Fig. 4 shows ΔC_ℓ well below cosmic variance, this is only a sanity check—not a guarantee of unbiased inference—because coherent same-sign residuals (even at the $\sim 0.1\%$ level for cosmic variace-limited data to ℓ) can bias parameters, so the proper validation is a posterior bias test rather than per-ℓ cosmic variance-limit comparisons. Real measurements further degrade per–mode precision –due for instance to finite $f_{\rm sky}$, beams, anisotropic noise, foregrounds, and systematics – experimental uncertainties per-multipole will be substantially larger in current and future experiments, making any residual numerical mismatch even less consequential.

5. Conclusions and future prospects

The extensive profiling we performed indicates that gCAMB achieves a speedup of a factor ~ 10 on an NVIDIA A100 compared with standard CAMB on 32 CPU cores. This substantially reduces the compute time required to generate typical emulator training sets of $10^{5–6}$ spectra. Validation tests also show that numerical differences introduced by the GPU port are fully negligible and remain well below the cosmic variance limit on full-sky at each multipole. This makes the adoption of gCAMB completely safe from numerical errors for next-generation surveys even at very high multipoles (small scales). Crucially, gCAMB is a feature-complete, drop-in replacement for CAMB, so migrating existing workflows typically requires very few changes. Moreover, the GPU port isolates only the line-of-sight projection, leaving the primordial and source-function modules unchanged—and therefore editable –as commonly done for CAMB– for testing specific theoretical models.

We therefore expect gCAMB to be useful to the community for producing large, high-accuracy spectral datasets more quickly, with reduced CPU hours consumption and improved sustainability. Indeed, a notable reduction in power consumption was observed in the high accuracy scenario, where the power consumption estimated by the gCAMB code is approximately half that of the CPU-only code. We note that further energy benchmarking on several different CPU and GPU models is needed, as this could depend on the specific hardware used here. As immediate next steps, porting also the DoNonFlatIntegration routine to the GPU should further increase throughput and make, in principle, systematic explorations of curved universes feasible and significantly cheaper. In addition, integrating gCAMB with sampling algorithms (e.g., MCMC) would enable direct evaluation of theoretical spectra without emulators when desired.

Among other possibilities, we envisage the following potential applications of gCAMB. Firstly, as anticipated, gCAMB is indeed well suited to producing massive datasets of extremely high-accuracy spectra to train robust power-spectrum emulators (Spurio Mancini et al., 2022; Jense et al., 2025). This is advantageous for standard ΛCDM and even more valuable for costlier non-flat cosmologies and extended models.

Models with sharp or resonant primordial features (e.g., axion–monodromy (Zeng et al., 2019) or EFT "standard clock" signals (Aslanyan et al., 2014; Calderon et al., 2025)) imprint rapid oscillations in $\Delta_\ell^X(k)$ and C_ℓ (see also Braglia et al., 2021). Computing every multipole ℓ without spline interpolation—and evaluating transfer kernels on dense k grids—improves fidelity in these scenarios. High-throughput GPU line-of-sight projections in gCAMB make such dense sampling practical for targeted searches in CMB and LSS.

Non-flat models replace spherical Bessel kernels with hyperspherical Bessel functions and are substantially more expensive (Lesgourgues and Tram, 2014; Kosowsky, 1998; Anselmi et al., 2023b,a; Tian et al., 2021). Efficient evaluation of curved-space transfer functions and line-of-sight projections is a known bottleneck; gCAMB can accelerate these steps, enabling systematic scans over $\Omega_K \neq 0$.

gCAMB could in principle be useful also for exploring anisotropic/topological and other beyond-FLRW signatures (Akrami et al., 2024). Statistical anisotropy or nontrivial topology couples modes and often demands fine sampling in both k and ℓ. Avoiding interpolation artifacts is particularly valuable when testing such models against large-angle CMB anomalies or anisotropic clustering statistics.

Acknowledgements

We thank Stefano Anselmi, Antony Lewis and Glenn Starkman for useful comments and discussion. We acknowledge the use of computing facilities provided by the INFN theory group (I.S. InDark) at CINECA.

This work has been supported by the Fondazione ICSC, Spoke 3 Astrophysics and Cosmos Observations. National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, PNRR) Project ID CN_00000013 "Italian Research Center on High-Performance Computing, Big Data and Quantum Computing" funded by MUR Missione 4 Componente 2 Investimento 1.4: Potenziamento strutture di ricerca e creazione di "campioni nazionali di R&S (M4C2-19)" - Next Generation EU (NGEU).

M.L. acknowledge the financial support from the INFN InDark initiative and from the COSMOS network through the ASI (Italian Space Agency) Grants 2016-24-H.0 and 2016-24-H.1-2018. M.L. is funded by the European Union (ERC, RELiCS, project number 101116027).

L. S. and N. A. acknowledge funding from "PaGUSci - Parallelization and GPU Porting of Scientific Codes" CUP: C53C22000350006 within the Cascading Call issued by Fondazione ICSC , Spoke 3 Astrophysics and Cosmos Observations.

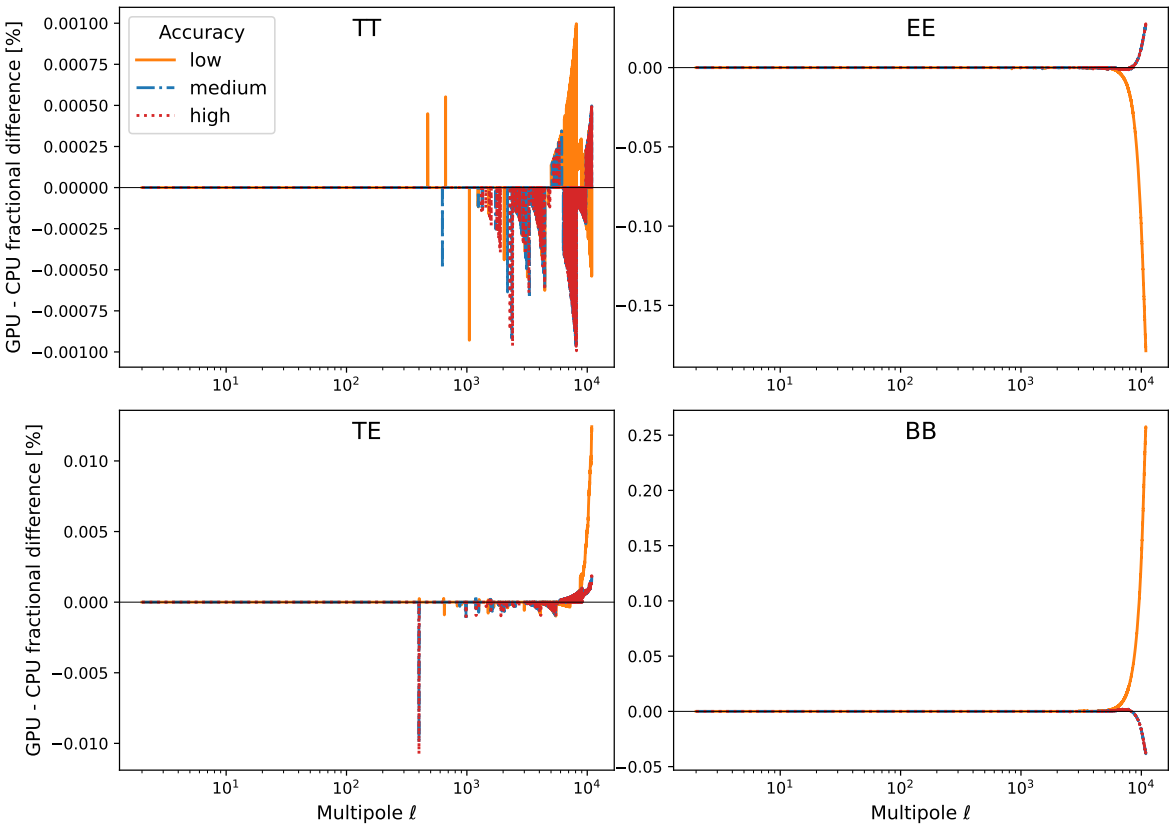


Figure 3: Fractional difference (in percentage) in low, medium and high accuracy spectra between `gCAMB` running on GPU and `CAMB` on CPUs.

References

Aghanim, N., et al. (Planck), 2020. Planck 2018 results. VI. Cosmological parameters. Astron. Astrophys. 641, A6. doi:10.1051/0004-6361/201833910, arXiv:1807.06209. [Erratum: Astron.Astrophys. 652, C4 (2021)].

Akrami, Y., et al. (COMPACT), 2024. Promise of Future Searches for Cosmic Topology. Phys. Rev. Lett. 132, 171501. doi:10.1103/PhysRevLett.132.171501, arXiv:2210.11426.

Albers, J., Fidler, C., Lesgourgues, J., Schöneberg, N., Torrado, J., 2019. CosmicNet. Part I. Physics-driven implementation of neural networks within Einstein-Boltzmann Solvers. JCAP 09, 028. doi:10.1088/1475-7516/2019/09/028, arXiv:1907.05764.

Anselmi, S., Carney, M.F., Giblin, J.T., Kumar, S., Mertens, J.B., O'Dwyer, M., Starkman, G.D., Tian, C., 2023a. What is flat ΛCDM, and may we choose it? JCAP 02, 049. doi:10.1088/1475-7516/2023/02/049, arXiv:2207.06547.

Anselmi, S., Starkman, G.D., Renzi, A., 2023b. Cosmological forecasts for future galaxy surveys with the linear point standard ruler: Toward consistent BAO analyses far from a fiducial cosmology. Phys. Rev. D 107, 123506. doi:10.1103/PhysRevD.107.123506, arXiv:2205.09098.

Aslanyan, G., Price, L.C., Abazajian, K.N., Easther, R., 2014. The Knotted Sky I: Planck constraints on the primordial power spectrum. JCAP 08, 052. doi:10.1088/1475-7516/2014/08/052, arXiv:1403.5849.

Assassi, V., Simonović, M., Zaldarriaga, M., 2017. Efficient evaluation of angular power spectra and bispectra. JCAP 11, 054. doi:10.1088/1475-7516/2017/11/054, arXiv:1705.05022.

Bolliet, B., Spurio Mancini, A., Hill, J.C., Madhavacheril, M., Jense, H.T., Calabrese, E., Dunkley, J., 2024. High-accuracy emulators for observables in ΛCDM, Neff, Σmν, and w cosmologies. Mon. Not. Roy. Astron. Soc. 531, 1351–1370. doi:10.1093/mnras/stae1201, arXiv:2303.01591.

Bonici, M., Bianchini, F., Ruiz-Zapatero, J., 2023. Capse.jl: efficient and auto-differentiable CMB power spectra emulation arXiv:2307.14339.

Braglia, M., Chen, X., Hazra, D.K., 2021. Comparing multi-field primordial feature models with the Planck data. JCAP 06, 005. doi:10.1088/1475-7516/2021/06/005, arXiv:2103.03025.

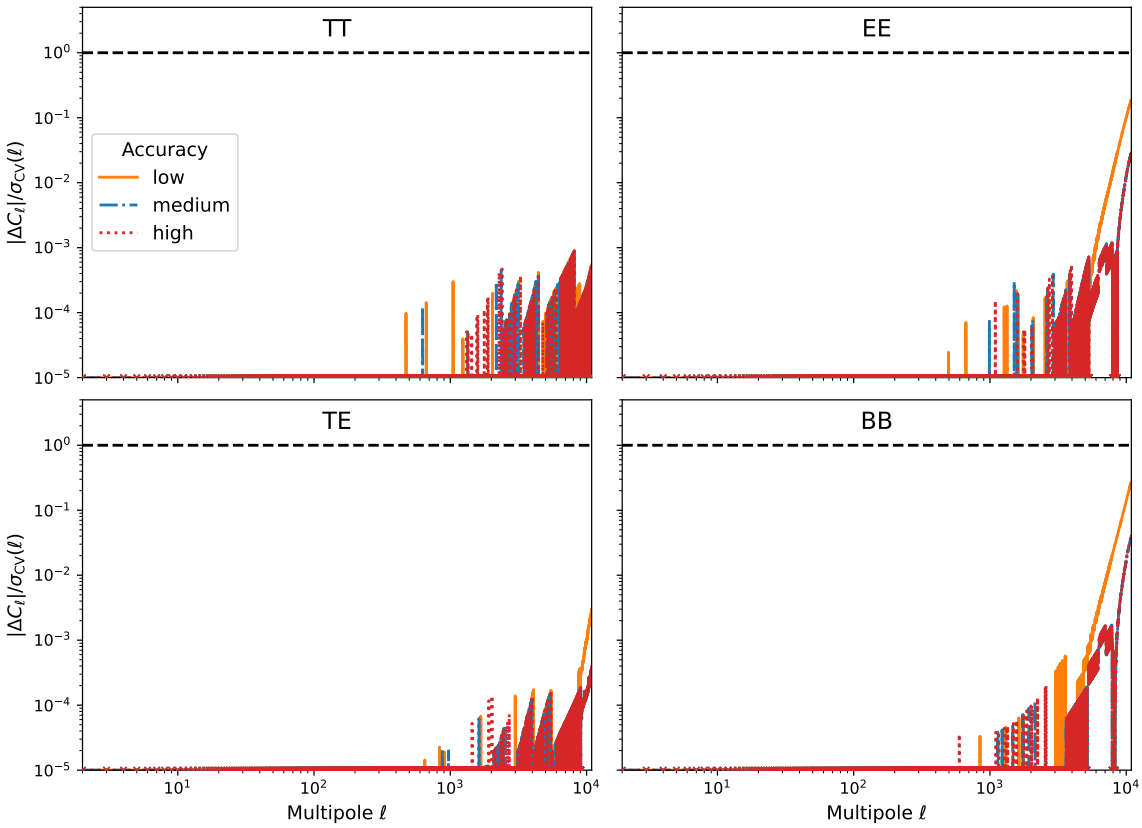


Figure 4: Ratio $|\Delta C_\ell|/\sigma_{\rm CV}(\ell)$ of the absolute value of the difference in low, medium and high accuracy spectra between `gCAMB` running on GPU and `CAMB` on CPUs and the cosmic variance computed on full-sky at each multipole. The dashed black horizontal line at 1 marks the strict per–multipole threshold given by cosmic variance. The numerical difference between `gCAMB` and `CAMB` is always below the cosmic variance at least by a factor 5.

Calderon, R., Simon, T., Shafieloo, A., Hazra, D.K., 2025. Primordial Features in light of the Effective Field Theory of Large Scale Structure `arXiv:2504.06183`.

Günther, S., Lesgourgues, J., Samaras, G., Schöneberg, N., Stadtmann, F., Fidler, C., Torrado, J., 2022. CosmicNet II: emulating extended cosmologies with efficient and accurate neural networks. JCAP 11, 035. doi:`10.1088/1475-7516/2022/11/035`, `arXiv:2207.05707`.

Henderson, S.W., et al., 2016. Advanced ACTPol Cryogenic Detector Arrays and Readout. J. Low Temp. Phys. 184, 772–779. doi:`10.1007/s10909-016-1575-z`, `arXiv:1510.02809`.

Howlett, C., Lewis, A., Hall, A., Challinor, A., 2012. CMB power spectrum parameter degeneracies in the era of precision cosmology. *JCAP* 1204, 027. doi:`10.1088/1475-7516/2012/04/027`, `arXiv:1201.3654`.

Jense, H.T., Harrison, I., Calabrese, E., Spurio Mancini, A., Bolliet, B., Dunkley, J., Hill, J.C., 2025. A complete framework for cosmological emulation and inference with CosmoPower. RAS Tech. Instrum. 4, rzaf002. doi:`10.1093/rasti/rzaf002`, `arXiv:2405.07903`.

Kosowsky, A., 1998. Efficient computation of hyperspherical bessel functions `arXiv:astro-ph/9805173`.

Laureijs, R., Amiaux, J., Arduini, S., Augueres, J.L., Brinchmann, J., Cole, R., Cropper, M., Dabin, C., Duvet, L., Ealet, A., Garilli, B., Gondoin, P., Guzzo, L., Hoar, J., Hoekstra, H., Holmes, R., Kitching, T., Maciaszek, T., Mellier, Y., Pasian, F., Percival, W., Rhodes, J., Saavedra Criado, G., Sauvage, M., Scaramella, R., Valenziano, L., Warren, S., Bender, R., Castander, F., Cimatti, A., Le Fèvre, O., Kurki-Suonio, H., Levi, M., Lilje, P., Meylan, G., Nichol, R., Pedersen, K., Popa, V., Rebolo Lopez, R., Rix, H.W., Rottgering, H., Zeilinger, W., Grupp, F., Hudelot, P., Massey, R., Meneghetti, M., Miller, L., Paltani, S., Paulin-Henriksson, S., Pires, S., Saxton, C., Schrabback, T., Seidel, G., Walsh, J., Aghanim, N., Amendola, L., Bartlett, J., Baccigalupi, C., Beaulieu, J.P., Benabed, K., Cuby, J.G., Elbaz, D., Fosalba, P., Gavazzi, G., Helmi, A., Hook, I., Irwin, M., Kneib, J.P., Kunz, M., Mannucci, F., Moscardini, L., Tao, C., Teyssier, R., Weller, J., Zamorani, G., Zapatero Osorio, M.R., Boulade, O., Foumond, J.J., Di Giorgio, A., Guttridge, P., James, A., Kemp, M., Martignac, J., Spencer, A., Walton, D., Blümchen, T., Bonoli, C., Bortoletto, F., Cerna, C., Corcione, L., Fabron, C., Jahnke, K., Ligori, S.,

Madrid, F., Martin, L., Morgante, G., Pamplona, T., Prieto, E., Riva, M., Toledo, R., Trifoglio, M., Zerbi, F., Abdalla, F., Douspis, M., Grenet, C., Borgani, S., Bouwens, R., Courbin, F., Delouis, J.M., Dubath, P., Fontana, A., Frailis, M., Grazian, A., Koppenhöfer, J., Mansutti, O., Melchior, M., Mignoli, M., Mohr, J., Neissner, C., Noddle, K., Poncet, M., Scodeggio, M., Serrano, S., Shane, N., Starck, J.L., Surace, C., Taylor, A., Verdoes-Kleijn, G., Vuerli, C., Williams, O.R., Zacchei, A., Altieri, B., Escudero Sanz, I., Kohley, R., Oosterbroek, T., Astier, P., Bacon, D., Bardelli, S., Baugh, C., Bellagamba, F., Benoist, C., Bianchi, D., Biviano, A., Branchini, E., Carbone, C., Cardone, V., Clements, D., Colombi, S., Conselice, C., Cresci, G., Deacon, N., Dunlop, J., Fedeli, C., Fontanot, F., Franzetti, P., Giocoli, C., Garcia-Bellido, J., Gow, J., Heavens, A., Hewett, P., Heymans, C., Holland, A., Huang, Z., Ilbert, O., Joachimi, B., Jennins, E., Kerins, E., Kiessling, A., Kirk, D., Kotak, R., Krause, O., Lahav, O., van Leeuwen, F., Lesgourgues, J., Lombardi, M., Magliocchetti, M., Maguire, K., Majerotto, E., Maoli, R., Marulli, F., Maurogordato, S., McCracken, H., McLure, R., Melchiorri, A., Merson, A., Moresco, M., Nonino, M., Norberg, P., Peacock, J., Pello, R., Penny, M., Pettorino, V., Di Porto, C., Pozzetti, L., Quercellini, C., Radovich, M., Rassat, A., Roche, N., Ronayette, S., Rossetti, E., 2011. Euclid Definition Study Report. arXiv e-prints , arXiv:1110.3193doi:10.48550/arXiv.1110.3193, arXiv:1110.3193.

Lesgourgues, J., 2011. The Cosmic Linear Anisotropy Solving System (CLASS) I: Overview. arXiv e-prints , arXiv:1104.2932doi:10.48550/arXiv.1104.2932, arXiv:1104.2932.

Lesgourgues, J., Tram, T., 2014. Fast and accurate CMB computations in non-flat FLRW universes. JCAP 09, 032. doi:10.1088/1475-7516/2014/09/032, arXiv:1312.2697.

Lewis, A., Challinor, A., . Camb: Code for anisotropies in the microwave background. GitHub repository. URL: https://github.com/lstorchi/CAMB. fork by Loriano Storchi. Original repository at https://github.com/cmbant/CAMB.

Lewis, A., Challinor, A., 2011. CAMB: Code for Anisotropies in the Microwave Background. Astrophysics Source Code Library, record ascl:1102.026. arXiv:1102.026.

Lewis, A., Challinor, A., Lasenby, A., 2000. Efficient computation of CMB anisotropies in closed FRW models. ApJ 538, 473–476. doi:10.1086/309179, arXiv:astro-ph/9911177.

Lifshitz, E., 1946. Republication of: On the gravitational stability of the expanding universe. J. Phys. (USSR) 10, 116. doi:10.1007/s10714-016-2165-8.

Lifshitz, E.M., Khalatnikov, I.M., 1963. Investigations in relativistic cosmology. Adv. Phys. 12, 185–249. doi:10.1080/00018736300101283.

Ma, C.P., Bertschinger, E., 1995. Cosmological perturbation theory in the synchronous and conformal Newtonian gauges. Astrophys. J. 455, 7–25. doi:10.1086/176550, arXiv:astro-ph/9506072.

de Melo, A.C., 2010. The new perf tool, in: Proceedings of the 2010 Linux Plumbers Conference. URL: https://www.kernel.org/doc/ols/2010/ols2010-pages-49-60.pdf.

NVIDIA Corporation, 2025a. NVIDIA Nsight Compute. https://developer.nvidia.com/nsight-compute. Accessed: 2025-08-26.

NVIDIA Corporation, 2025b. NVIDIA System Management Interface (nvidia-smi). https://developer.nvidia.com/nvidia-system-management-interface. Accessed: 2025-08-26.

Peebles, P.J., 1980. The Large-Scale Structure of the Universe. Princeton University Press.

Schöneberg, N., Simonović, M., Lesgourgues, J., Zaldarriaga, M., 2018. Beyond the traditional Line-of-Sight approach of cosmological angular statistics. JCAP 10, 047. doi:10.1088/1475-7516/2018/10/047, arXiv:1807.09540.

Seljak, U., Zaldarriaga, M., 1996. A Line of sight integration approach to cosmic microwave background anisotropies. Astrophys. J. 469, 437–444. doi:10.1086/177793, arXiv:astro-ph/9603033.

Simons Observatory (Simons Observatory), 2019. The Simons Observatory: Science goals and forecasts. JCAP 02, 056. doi:10.1088/1475-7516/2019/02/056, arXiv:1808.07445.

Spurio Mancini, A., Piras, D., Alsing, J., Joachimi, B., Hobson, M.P., 2022. CosmoPower: emulating cosmological power spectra for accelerated Bayesian inference from next-generation surveys. Mon. Not. Roy. Astron. Soc. 511, 1771–1788. doi:10.1093/mnras/stac064, arXiv:2106.03846.

Storchi, L., a. Camb gpuport branch. GitHub repository. URL: https://github.com/lstorchi/CAMB/tree/gpuport, doi:10.5281/zenodo.17035660.

Storchi, L., b. forutils gpuport branch. GitHub repository. URL: https://github.com/lstorchi/forutils/tree/gpuport, doi:10.5281/zenodo.17035562.

Tian, C., Anselmi, S., Carney, M.F., Giblin, J.T., Mertens, J.B., Starkman, G., 2021. Question of measuring spatial curvature in an inhomogeneous universe. Phys. Rev. D 103, 083513. doi:10.1103/PhysRevD.103.083513, arXiv:2010.07274.

Turisini, M., Amati, G., Cestari, M., 2023. Leonardo: A pan-european pre-exascale supercomputer for hpc and ai applications. URL: https://arxiv.org/abs/2307.16885, arXiv:2307.16885.

Weinberg, S., 1972. Gravitation and Cosmology: Principles and Applications of the General Theory of Relativity. John Wiley and Sons, New York.

Wienke, S., Springer, P., Terboven, C., an Mey, D., 2012. Openacc — first experiences with real-world applications, in: Euro-Par 2012 Parallel Processing, Springer. pp. 859–862. doi:10.1007/978-3-642-32820-6_85.

Zeng, C., Kovetz, E.D., Chen, X., Gong, Y., Muñoz, J.B., Kamionkowski, M., 2019. Searching for Oscillations in the Primordial Power Spectrum with CMB and LSS Data. Phys. Rev. D 99, 043517. doi:10.1103/PhysRevD.99.043517, arXiv:1812.05105.