3rd INTERNATIONAL WORKSHOP ON AI + ASTRONOMY: MODELS, DATA, DISCOVERY

Code-specialized LLMs in HPC

A case study in radio-astronomy pipeline acceleration



VENUE & DATE
Hangzhou
2026

PRESENTED BY
Hamza Chouh
EPFL

École polytechnique fédérale de Lausanne

Image: ESO/Y. Beletsky (CC BY 4.0)

Quick introducation

Where am I talking from?

- Member of SCITAS (High Performance Computing service provider) at EPFL
- HPC (not AI) software engineer, not an astronomer

EPFL SCITAS
Scientific IT and Application Support

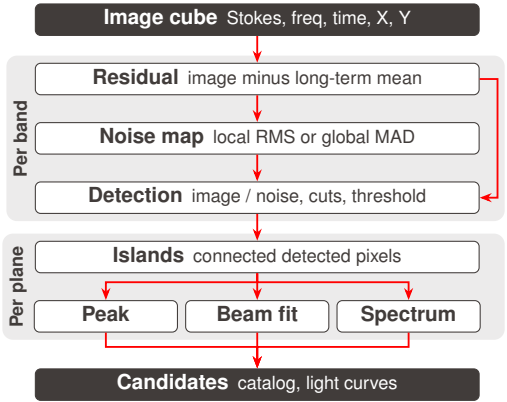
→ My perspective on the impact of LLMs in HPC

- Situation: Python code with no reference paper, going from theory to implementation was not easy
- Ideas:
  - Try to leverage code-specialized LLMs (here, Claude Code) to accelerate the process
  - Save time by using some code generation and text analysis features
  - Make the process reproducible by recording the steps, the hypotheses and the results

Breifast

Transient detection pipeline

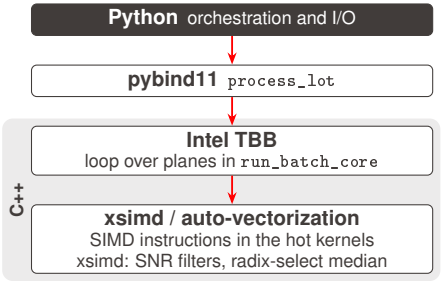
- Transient detection pipeline for radio-interferometry targeting MeerKAT
- Written in Python by the RATT group (Rhodes University, South Africa)
- Currently not optimized for HPC
- Aim of this work: HPC-optimized C++ implementation on CPU and GPU



C++ implementation

What changed in the code

- C++ kernels in a separate module, Python orchestration through `pybind11` links
- Parallel execution on CPU and enforced vectorization using `xsimd`
- Rethought implementation, improved memory access patterns and merged loops
- Alternative implementations of some algorithms, e.g. median filtering



Main result

Substantial speedup and identical results

- Overall, 24× acceleration on CPU
- About 42% scaling efficient on 72 cores
- Code mostly memory-bound
- Identical results to the original Python implementation
- Automated testing and benchmarking to keep track of performance, ensure correctness, robustness and reproducibility

Bench lot
1080 planes (40 GB), 72 threads

	Time	Acceleration
Python	101.89 s	×1
Initial C++	24.63 s	**×4.1**
Optimized C++	4.26 s	**×24**

Time per run, acceleration against Python
Same node, mean over all iterations

Attempt 1: codeless approach

From Python to C++ with a code-specialized LLM

First easy attempt:

- Hand the Python code to the LLM and ask for a C++ translation
- We used Claude Code for code generation
- Result:
  - A C++ translation that seemed correct and reasonably well written, mimicking Python code with `xtensor` calls
  - Tests: unit, integration, end-to-end
  - Scaling was poor and parallelization attempts made by the LLM were not efficient (wrong level of parallelism)

Worst thing: absolute loss of mastery over the code. No control whatsoever.

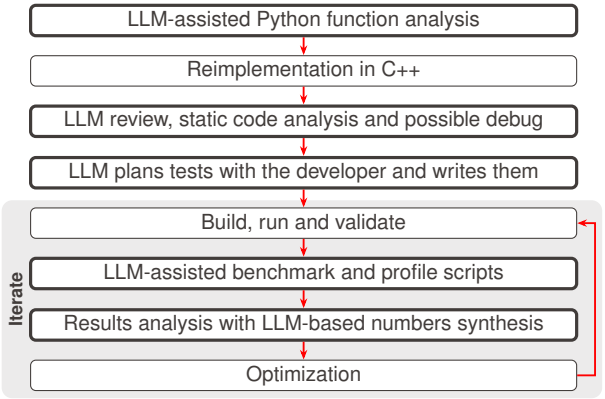
Attempt 2: peer programming

Efficiently saving time and reducing errors

Two options:

1. Spend hours reviewing a fully generated code
2. Spend hours rewriting the code with the LLM as an assistant

Option 2 has the benefit of integrating the algorithms learning processes and naturally builds mastery over the code through practice.



LLM as a reference paper

From implementation to theory, and from theory to implementation

- LLM does static input code analysis (here, Python) backed by general knowledge in radio-astronomy
- It can also make small scripts and tests to verify claims
- Provides an easier way for the developer to understand the code
- Particularly useful for HPC engineers who are not specialized in astronomy
- Saves a lot of time when not working directly with the code developers

LLM as a reference paper

From implementation to theory, and from theory to implementation

Python: `max_to_std_ratio(n)` **in** `breifast.py`**, with what each line computes**

```
x = numpy.linspace(-10, 10, 1000)
f = 0.5 * (1.0 + scipy.special.erf(x / numpy.sqrt(2.0)))
F = 1.0 - (1.0 - f)**n
ratio = numpy.abs(numpy.interp(0.5, F, x))
```

→ Levels to test, from -10σ to $+10\sigma$
→ Chance that one noise pixel is below x
→ Chance that at least one of n is below x
→ Level where that chance is 50%, called r_n

LLM explains

Formula $\Phi(r_n)^n = \frac{1}{2}$: half of the time, at least one of the n noise pixels is brighter than $r_n\,\sigma$.

Used as $\hat{\sigma} = \max|\mathrm{stamp}| \,/\, r_n$ for the small image around each candidate, with the candidate itself masked out.

Context A 401×401 stamp has 160 801 pixels, so noise alone reaches $4.45\,\sigma$ in half of the stamps ($5.26\,\sigma$ over a whole 3072×3072 plane). At 5σ noise gives about 5 false pixels per plane, at 6σ only 0.02.

LLM as a linter

Review of developer-written code

- LLM works as a static code analyzer on developer-written code
- Spots typos, obvious mistakes and possible higher-level mistakes
- Can also make implementation suggestions and variable-level code review

Developer-written code (sliding-window RMS pass)

```
accumulators(i_y) = accumulator;  // one value per row
```

The seed loop has the accumulator indexed by row instead of by column.

`accumulators[i]` has to be the running vertical sum for column `block_start + i`, accumulated across rows. [...]

As written it also writes out of bounds whenever `max_y_of(0) > block_width`, which `boxsize > 1024` would reach.

Review comment by Claude Code, 24 Sep 2026

LLM-assisted profiling

Scripts generation, results analysis and reproducibility

- Using LLM to generate VTune custom scripts with environment management and configuration
- The LLM helps post-process results and explore the results database
- Keeps track of all scripts, attempts and results, the rigor most people skip when focused on the work

Prompt, 16 Sep 2026

create another script to run a threading analysis using VTune

Generated script — **Claude Code**

```
vtune -collect threading -start-paused \
  -knob sampling-and-waits=hw \
  -knob sampling-interval=10 \
  -result-dir "$RESULT_DIR" \
  -- .../builddir/bench .../lot_diff36_cpp/ \
       "$NUM_THREADS" "$ITERATIONS"
```

Claude Code first read `vtune -help collect threading` for the valid knobs. The script also records `provenance.txt` (git head, build flags) and `source.patch` (uncommitted diff).

Profiling examples

Database queries, scripts and improvement ideas

- Explores the VTune results database and ties samples back to source lines
- Provides complementary information to Intel VTune's interface
- Pins down performance issues and suggests improvements

Prompt, 4 Sep 2026

Look at that [VTune result] And tell me which xtensor calls result in all these accessor time samples

```
SELECT f.name, COUNT(*) AS cnt
FROM dd_sample s JOIN dd_callsite c ON s.callsite = c.rowid
JOIN dd_code_location cl ON c.code_loc = cl.rowid
JOIN dd_source_location sl ON cl.src_loc = sl.rowid
JOIN dd_function f ON sl.function = f.rowid
GROUP BY f.rowid ORDER BY cnt DESC
```

Then the call chain of each sample, ranked by share:

1. `xt::median(xt::abs(plane))`, line 307, 17%: a lazy expression that falls back to the generic strided iterator
2. `compute_snr_and_apply_filters`, loop at line 142, 11%

Profiling examples

Database queries, scripts and improvement ideas

- Explores the VTune results database and ties samples back to source lines
- Provides complementary information to Intel VTune's interface
- Pins down performance issues and suggests improvements

Prompt, 16 Sep 2026

> The last run [VTune result] shows one line with very high sync wait time but no stack info. Are you able to interpret it

```
vtune -report summary ...
vtune -report hotspots -group-by thread -format=csv ...
```

`[Outside any known module]` is the VTune bucket for time that threads spent not running: 2834 s of inactive sync wait (idle TBB workers) and 610 s of preemption. No instruction was executing, so there is no stack to show.

Some examples of LLM failures

Confident claims that measurements overturned

LLM claim	What happened
Claimed parallelism was not possible to improve after first implementation	Changing the level of parallelization and reorganizing the loop proved it wrong
“Modest but real wall-clock improvement” for 11.711 → 11.387 s (2.8%)	Within measurement noise, although UPI load fell to 0%
Low thread utilization comes from task granularity or load imbalance	700 ms per task is long enough. The low reading was a profiler artifact
Not a DRAM-bandwidth problem (VTune: 0.1% bound)	Shared memory bandwidth among threads is the main limitation
Tiling is a good strategy to improve cache locality	No improvement (+0.4% at 72 threads)

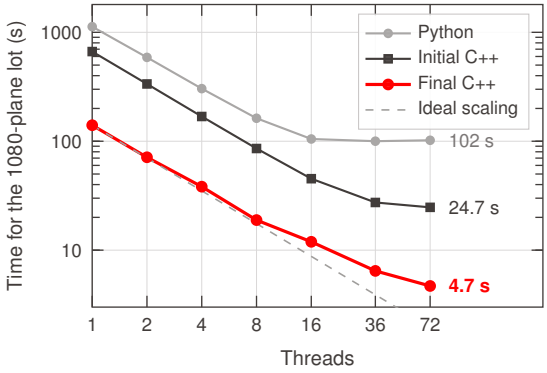
Performance improvements

Examples of LLM-assisted optimizations

What the LLM did	Change made	Gain at 72 threads
Traced VTune samples back to `xt::median(xt::abs(plane))`	Replaced `xt::median` with `std::nth_element`	**−18%**
Proposed an exact radix select for the medians	Two read-only histogram passes, no 37 MB copy (113 → 31 ms per call)	**−25%**
Noticed that -O2 runs no vectorizer and assertions were on	Release build: -O3 -march=icelake-server (−19% on one thread)	**−11%**
Showed that the SNR loop was compute-bound	SNR filter on xsimd batches (−23% on one thread)	**−12%**
Ranked the next SIMD candidates and reviewed the xsimd code	Radix select on xsimd batches (−17% on one thread)	**−6%**

Current performance results

Python, initial C++ and optimized C++



	1 thread	72 threads	Speedup
Python	1127 s	102 s	×11 (15%)
Initial C++	666 s	24.7 s	×27 (37%)
Final C++	140 s	4.69 s	**×30** (42%)

Configuration. The bench lot, 36 planes replicated 30 times: 1080 planes of 3072 × 3072 float32 (40 GB). Two 36-core Xeon Platinum 8360Y (Ice Lake, AVX-512), one TBB arena, data on `/scratch`, exclusive node. Means over all iterations.

Current performance results

Python, initial C++ and optimized C++

Topic	Where the CPU version stands
SIMD	xsimd in the SNR filter (−12% at 72 threads) and the radix select (−6%), release build for AVX-512. Still scalar: the histogram scatters and loops that are already memory bound.
Memory bound	DRAM traffic averages 147 GB/s and peaks at 363 GB/s, against about 400 GB/s for the node (STREAM triad: 300 GB/s). Half of the accesses are remote, and CPU time per item grows by 58% from 1 to 72 threads.
GPU outlook	An H100 reads its HBM at 2.25 TB/s, 6 to 8 times the node, and its 60 MiB L2 holds a whole 37 MB plane. PCIe becomes the limit: 40.7 GB at 46 to 52 GB/s takes 0.8 to 0.9 s, so at best about 5 times faster than the CPU. GPU port under way, no timing yet.

Repo as reproducible research

- LLM-based automated scripting and process documentation for reproducible research
- Built history of optimization process
- Significant improvement in terms of code quality and accessibility

`optimization/step_12/README.md`, **condensed**

Hypothesis	An exact radix select, two 16-bit histogram passes over an order-preserving key, removes the 37 MB copy made by `nth_element`
Check	6000 random arrays give the same median as `nth_element`
Result	113 → 31 ms per median. Bench lot at 72 threads: 10.41 → 7.84 s (−25%)
Reproduce	`git checkout perf-step_12` `sbatch` TODO script and job id

17
optimization steps, one README each

33
SLURM scripts, one question each

74
Catch2 tests on the C++ kernels

Conclusions

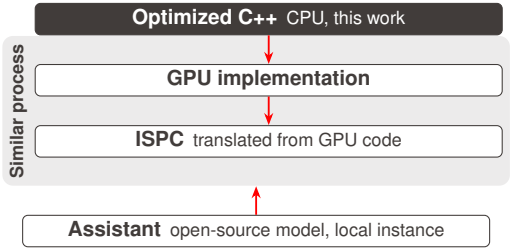
- Significant acceleration of a radio-astronomy transient detection kernel
- Gain in code quality, efficiency and reproducibility
- Keeping control over the code during the whole process
- Models used: Claude Sonnet 5 and 5.5, Opus 5 and 5.5
- No frontier model used, which could have proven some of our conclusions wrong

Delegate	Peer code
The LLM writes, the human reviews	The human writes, the LLM explains and reviews
Benchmark and profiling scripts	Core kernels
Tables, summaries, log analysis	Optimization choices
Profile database queries	Experiment design
Tests, plan approved first	Reading the measurements
Prototypes and throwaway code	

Perspective

Upcoming work

- GPU implementation
- Keeping a similar development process
- Experimenting using CUDA to ISPC translation
- Switching to open-source models on a local instance



Food for thought

How does this impact my job?

- Our first attempt proved inconclusive due to the role it left for the developers
- Rethinking our own role in engineering: offload grunt work, keep the fun?
- Is it only yet another level of automatization?
- Can we leverage the generalized access to these tools to make this automatization step fruitful for the many?

−17%



Comprehension quiz score with AI help. Participants who asked for explanations scored higher

Anthropic randomized trial, Jan 2026, n = 52

+19%



Task time for experienced developers with AI, who believed they were 20% faster

METR randomized trial, Jul 2025

Weaker



Brain connectivity measured by EEG when writing essays with an LLM

MIT Media Lab, Jun 2025, n = 54, preprint

3rd INTERNATIONAL WORKSHOP ON AI + ASTRONOMY: MODELS, DATA, DISCOVERY

Questions?



CONTACT
hamza.chouh@epfl.ch

PRESENTED BY
Hamza Chouh
EPFL

École polytechnique fédérale de Lausanne