StrasGPT Parallelization Project (2025–2026)

Parallel Programming (MSc)

Purpose and Scope

This project requires students to parallelize a real Large Language Model (LLM) inference engine written in C, namely **StrasGPT**, a direct implementation of the LLaMa 3.x transformer architecture. The objectives are twofold: (i) accelerate inference on a single node using **OpenMP**; and (ii) enable multi-process (and potentially multi-node) execution using **MPI**, including support for models that exceed the memory capacity of a single node. All computations must remain in *float32* precision.

1 Execution Environment and Target Models

- **Hardware:** University Virtual Machines (VMs), each with **16 GB RAM**. *All* reported performance numbers must be collected on these VMs. Plan experiments early to avoid end-of-term congestion.
- **Models:** Llama-3.2-1B, Llama-3.2-3B, Llama-3.1-8B, and Llama-3.1-70B.
- **Precision:** All computations must be **float32**. Quantization or computation skipping is not permitted.

2 Provided Codebase

StrasGPT is a single-binary C implementation compatible with HuggingFace safetensors. The repository layout includes: `include/`, `source/`, `assets/`, and `test/prompts/`. Build targets:

- `make` — sequential build (baseline).
- `make parallel` — OpenMP+MPI build (initially prints; you will implement real parallelism).
- `make debug`, `make asan` — debugging and AddressSanitizer builds.

3 Obtaining Models

On the University VMs, small models are provided in `~/partage/`. Outside the VMs, download from the University Seafile (for the small models) or from HuggingFace (requires account and access token). For evaluation and reporting, restrict experiments to the four target models listed in Section 1.

4 Running the Baseline

An example sequential invocation is:

```
./strasgpt -m <path_to_model_dir> -p "Once␣upon␣a␣time" -n 17
```

The program prints two key metrics at the end of execution:

- **Prompt processing (prefill)**: throughput while analyzing the input and producing the first token.
- **Token generation (decode)**: throughput while generating subsequent tokens.

These two metrics are the primary performance indicators for this project.

5 Functional Requirements

Your final parallel version **must** satisfy the following:

(a) **Clean build:** compiles without errors or warnings under the `make parallel` target (we recommend enabling `-Werror`).

(b) **OpenMP control:** accepts `-n <threads>` to set the OpenMP thread count, and actually calls `omp_set_num_threads` (or equivalent).

(c) **MPI execution:** runs correctly under `mpirun -n <processes> ./strasgpt ...`.

(d) **Metrics:** prints prefill and decode performance lines at the end, matching the baseline format.

(e) **Precision:** keeps float32 throughout; no reduced-precision shortcuts; no computation skipping.

CLI note. If the original code uses `-n` to denote the number of generated tokens, you must introduce a distinct option (e.g., `-k <steps>` or `-s <steps>`) and document it. In the final deliverable, `-n` *must* set OpenMP threads.

6 Code Entry Points

Modify only the small, well-identified parts of the code:

- `source/strasgpt.c`: `main()` — initialization, option parsing, and generation loop.
- `source/transformer.c`: `transformer_predict()` and especially `transformer_predict_chunk()` — computational hotspots to parallelize.
- `source/transformer.c`: `weights_from_safetensors()` — tensor allocation and weight loading; may be adapted for sharded loading in MPI.

7 Parallelization Requirements

7.1 OpenMP (intra-node)

Your OpenMP implementation should exploit parallelism over attention heads, hidden dimensions, and token positions. Suggested targets include: embedding gathers, RMSNorm, Q/K/V projec-

tions, attention score computation and softmax (row-wise), value aggregation, and feed-forward layers. Keep inner loops vectorizable and outer loops parallel.

7.2 MPI (inter-process)

Your MPI implementation must support at least one sensible strategy, such as:

- **Layer pipeline (model parallelism):** split transformer layers across ranks; pass activations forward using point-to-point communication.
- **Head/tensor sharding:** split attention heads and corresponding weight slices across ranks; combine partial results using collectives (e.g., `MPI_Allreduce`).
- **Vocabulary/logits sharding (starter approach):** each rank computes a slice of the output logits; combine with `MPI_Allgatherv`; sample on rank 0; broadcast the next token.

For very large models, sharded weight loading (e.g., embedding and output layers) is encouraged to fit memory limits. A hybrid MPI+OpenMP configuration is allowed; avoid excessive thread-level MPI unless justified.

8 Performance Evaluation and Reporting

All measurements must be collected on the University VMs.

Workloads

- **Prefill-dominant**: long prompts (e.g., 1–2k tokens), generate 1–8 tokens.
- **Decode-dominant**: short prompts (e.g., 8–32 tokens), generate 64–256 tokens.

Scalings

- **OpenMP scaling**: `-n 1,2,4,8,...` (subject to VM core count).
- **MPI scaling**: `mpirun -n 2,4,...` within memory limits. For hybrid runs, report (processes × threads).

Metrics and Formulas

Record, for both prefill and decode:

- tokens processed/generated, elapsed time, and tokens/s.
- **Speedup:** $S_p = T_1/T_p$.
- **Efficiency:** $E_p = S_p/p$ (state whether p is threads, processes, or their product for hybrid runs).

Provide clear plots (tokens/s vs. threads/processes). Briefly analyze scaling behavior (memory bandwidth limitations, softmax reductions, collective costs, pipeline bubbles) and model-size effects.

9 Deliverables

(1) Source Code

- The parallel implementation, built by `make parallel` with no warnings.
- A short build/run guide (README/BUILD) explaining the CLI and any environment variables (e.g., `OMP_PLACES`, `OMP_PROC_BIND`).

(2) Report (PDF)

A single PDF named `rapport_LASTNAME1_LASTNAME2.pdf` including:

1. Student names and GitLab usernames.
2. Design overview (OpenMP and/or MPI) and rationale.
3. Implementation notes: modified parts, synchronization, data layout changes.
4. Performance results on VMs with tables and plots (prefill & decode), speedup and efficiency.
5. Per-model conclusions for Llama-3.2-1B, 3B; Llama-3.1-8B, 70B (prompt processing and token generation).
6. Difficulties and lessons learned; potential future work (optional).

If time is limited, focusing on a single aspect (OpenMP *or* MPI) is acceptable, provided the scope and limitations are clearly reported.

10 Rules and Constraints

- **Precision**: float32 only; no quantization or pruning/skip tricks.
- **Frameworks**: OpenMP for multithreading and MPI for multiprocessing only (no CUDA/OpenCL/TBB, etc.).
- **Code scope**: limit modifications to designated hot spots unless well-justified and documented.
- **Reproducibility**: list exact command lines and environment; note that text generation may vary due to floating-point order.
- **Privacy**: keep the repository private; share only within the binôme and instructors upon request.

11 Grading and Bonus

Grading emphasizes correctness of the parallel execution, code quality, clarity of design, and rigor of experimental methodology. A bonus is awarded for the highest demonstrated acceleration under fair and reproducible VM conditions.

12 Academic Integrity

Collaboration is restricted to your binôme. Do not share code across groups. Cite any external ideas used. Do not copy code from other parallel projects.

Pre-Submission Checklist

- `make parallel` builds without warnings (consider `-Werror`).
- `-n <threads>` controls OpenMP; `mpirun -n <processes>` runs.
- Prefill and decode metrics are printed.
- Report PDF includes plots, speedup, efficiency, and per-model conclusions.
- Repository is private; README documents build/run.

Appendix: Example Commands

Baseline run.

```
./strasgpt -m ../model_zoo/Llama-3.2-1B/ -p "Once␣upon␣a␣time" -k 64
```

OpenMP scaling.

```
./strasgpt -m ../model_zoo/Llama-3.2-1B/ -f test/prompts/goldilocks_32.
    txt -k 128 -n 1
./strasgpt -m ../model_zoo/Llama-3.2-1B/ -f test/prompts/goldilocks_32.
    txt -k 128 -n 8
```

MPI (hybrid example).

```
mpirun -n 4 ./strasgpt -m ../model_zoo/Llama-3.2-1B/ -f test/prompts/
    goldilocks_8.txt -k 128 -n 4
```

Good luck, and enjoy racing for the highest tokens per second!