HW2 Report

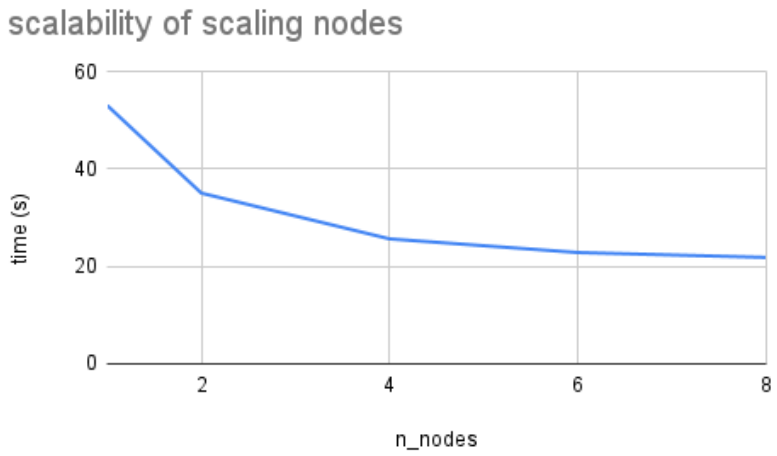
R13922213 梁正

1. Implementation

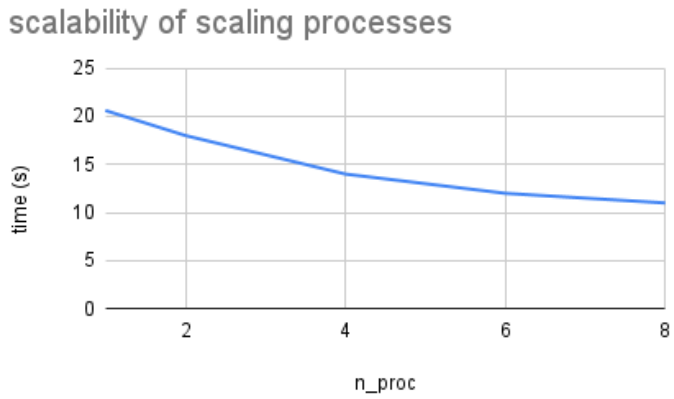
- Task partitioning
  - Image-level decomposition: work is partitioned by groups of pixels / regions of the image so each worker computes gradients and local responses on a disjoint image tile. This reduces cross-worker memory contention and keeps most data local to a worker.
  - Keypoint-level decomposition: after candidate keypoints are detected, descriptor computation is parallelized by assigning disjoint subsets of keypoints to workers. Descriptor computation is independent per keypoint, so this is an embarrassingly parallel subtask.
  - Hybrid approach: coarse-grained image tiling is used where spatial locality matters (convolution, gradient), and fine-grained per-keypoint parallelism is used for descriptor and matching phases.
- Scheduling algorithm
  - Static partitioning for spatial tasks: for convolution and gradient stages, a static block partition (horizontal strips or tiles) is used to minimize synchronization and to maximize cache reuse.
  - Dynamic scheduling for keypoint/descriptors: descriptor workloads vary per keypoint (different neighborhood sizes, boundary checks). For these loops an OpenMP-like dynamic schedule (small chunk size) is used to balance load across workers and avoid idle threads when some keypoints are more expensive.
  - If running on multiple nodes/processes, a simple work distribution (equal number of tiles or equal number of keypoints per rank) is used at launch time; intra-node threads use the dynamic schedule.
- Techniques to reduce execution time
  - Reduce memory traffic:
  - Work on contiguous data slices to improve cache locality.
  - Use per-thread temporary buffers to avoid frequent heap allocation and to eliminate false sharing.
  - Minimize synchronization:
  - Avoid global locks; aggregate results per-thread and do a single reduction step at the end.
  - Use atomic operations only when absolutely necessary.
  - Insert timing probes around major phases to find bottlenecks.

- Other efforts / engineering choices
  - oundary handling: explicitly handle image borders to avoid branching/conditionals inside inner loops.
  - Communication: use MPI_AllGatherv for fine-grained collective operaion
- Difficulties encountered and solutions
  - Load imbalance: descriptor computation varied per keypoint causing some workers to finish early. Solved by switching keypoint loops to dynamic scheduling with small chunk sizes so slow keypoints are redistributed.
  - Race conditions / false sharing: initial implementation used shared buffers and suffered inconsistent results and poor scaling. Solved by giving each thread its own temporary buffers and doing a final merge (reduction) step.
  - Memory bandwidth limits: scaling stalled beyond a certain number of cores. Mitigations included improving locality (tile processing), minimizing redundant reads, and reducing temporary allocations.
  - Border and numerical issues: descriptor bins near image border required careful handling. Solved by explicit border checks and by clamping / padding neighborhoods rather than adding conditional branching in inner loops.
  - Debugging parallel bugs: used incremental testing — first single-thread correctness tests (validate), then small-thread runs, then profiling to find hotspots; added assertion checks in debug builds to catch invariants.

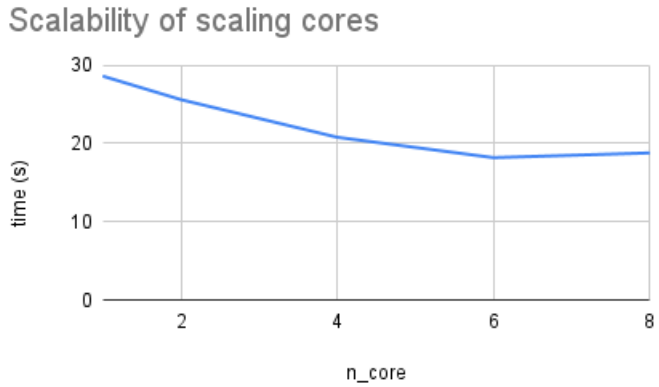
2. Analysis (on TID 06)
   - Scalability of scaling nodes



- Scalability of scaling processes



- Scalability of scaling cores



3. Conclusion

   This assignment reinforced that measured optimization wins: use performance profiling to find hotspots before changing code, then apply appropriate parallelization patterns (data parallelism, task/pipeline decomposition, and hybrid MPI+threads) to maximize concurrency while minimizing synchronization and communication.