SIFT Parallelization Report

Parallelization Strategy

This implementation uses a **hybrid MPI+OpenMP approach** with spatial domain decomposition to parallelize the SIFT (Scale-Invariant Feature Transform) algorithm.

Architecture Overview

The parallelization strategy divides the computational workload across multiple MPI ranks using a 2D Cartesian grid topology, with each rank processing a spatial tile of the image. Within each rank, OpenMP threads are used to further parallelize computationally intensive loops.

Phase-by-Phase Strategy

1. Gaussian Pyramid Generation (`generate_gaussian_pyramid_parallel`)

- **Tile-based decomposition**: Image divided into tiles across MPI ranks using Cartesian grid topology
- **Parallel Gaussian blur** (`gaussian_blur_parallel`):
  - Each rank blurs its local tile independently
  - Halo exchange operations for boundary pixels between neighboring ranks
  - Separable convolution: vertical pass → halo exchange → horizontal pass → halo exchange
  - OpenMP parallelization within each pass for interior pixels
- **Adaptive strategy**: Small octaves (when tiles become too small) processed sequentially on rank 0 to avoid communication overhead

2. DoG Pyramid (`generate_dog_pyramid_parallel`)

- **Embarrassingly parallel**: Each rank computes pixel-wise differences on its local tiles independently
- No inter-rank communication required
- OpenMP parallelization for difference computation

3. Gradient Pyramid (`generate_gradient_pyramid_parallel`)

- **Distributed computation**: Each rank computes gradients on local tiles
- **Halo exchange**: Required for accurate gradient computation at tile boundaries
- **OpenMP parallelization**: Interior pixels processed with `#pragma omp parallel for`

4. Keypoint Detection (`find_keypoints_parallel`)

- **Interior ownership model**: Each rank detects keypoints in its tile using a 1-pixel interior boundary rule
- **Halo-aware extrema checking**: Boundary pixels checked using exchanged halo data from neighboring ranks
- **Local refinement**: Keypoint refinement performed locally on each rank
- **Gather operation**: All detected keypoints gathered to rank 0

5. Orientation & Descriptor (`find_keypoints_and_descriptors_parallel`)

- **Gather-then-compute approach**:
  - Gradient pyramids and keypoints gathered to rank 0
  - Rank 0 performs orientation assignment and descriptor computation
- **OpenMP parallelization**: Keypoint processing parallelized with dynamic scheduling across threads
- **Trade-off**: Centralized computation avoids complex distributed orientation/descriptor logic

---

Performance Analysis

Sequential Baseline (1 rank, 1 thread)

- **Total execution time**: 127,817.89 ms (~128 seconds)
- **Major hotspots**:
  - Gaussian pyramid generation: 61,010 ms (47.6%)
  - Orientation & descriptor computation: 35,798 ms (27.9%)
  - Gradient pyramid generation: 18,454 ms (14.4%)
  - Keypoint detection: 5,556 ms (4.3%)
  - DoG pyramid generation: 2,598 ms (2.0%)

Parallel Version (4 ranks, 16 threads)

- **Total execution time (max across ranks)**: 39,241.50 ms (~39 seconds)
- **Overall speedup**: **3.26×**
- **Parallel efficiency**: 81.5% (3.26/4)

Detailed Phase Breakdown

Phase	Sequential (ms)	Parallel Max (ms)	Speedup	Efficiency
Gaussian pyramid	61,010	19,229	**3.17×**	79.3%
DoG pyramid	2,598	1,950	1.33×	33.3%
Gradient pyramid	18,454	4,518	**4.08×**	102.0%
Keypoint detection	5,556	2,298	2.42×	60.5%
Orientation/descriptor	35,798	3,265	**10.96×**	68.5%*

*Using 16 OpenMP threads on rank 0

Communication Overhead Analysis

Total MPI communication time: 29,238.74 ms (36.4% of parallel runtime)

Key communication costs:

- `gather_gradient_pyramid`: 8,074 ms per rank
  - Transfers entire gradient pyramid to rank 0
  - Largest communication bottleneck

- `MPI_Bcast_base_dimensions`: Up to 9,731 ms
  - Includes synchronization overhead
  - Broadcasting base image dimensions for each octave
- **Halo exchanges**: ~441 ms total
  - Relatively efficient for boundary data exchange
  - Small overhead compared to computation
- `gather_keypoints`: ~14 ms per rank
  - Minimal cost due to small keypoint data size

Load Balancing

- **Max time**: 39,241.50 ms (slowest rank)
- **Min time**: 35,955.72 ms (fastest rank)
- **Load imbalance**: 9.1%

Sources of imbalance:

1. Rank 0 handles small octaves exclusively (`prepare_octave_bases`: 9,703 ms)
2. Non-uniform keypoint distribution may cause slight variation in detection time
3. Rank 0 performs all orientation/descriptor computation (3,265 ms exclusive work)

Key Findings

Strengths ✅

1. **Excellent OpenMP scaling**: Orientation/descriptor phase achieves 10.96× speedup with 16 threads
2. **Effective domain decomposition**: Gradient pyramid shows 4.08× speedup with 4 ranks
3. **Efficient halo exchange**: Communication overhead for boundary data is minimal (~441 ms)
4. **Good overall speedup**: 3.26× speedup on 4 ranks demonstrates effective parallelization

Bottlenecks ⚠️

1. **Communication-bound**: 36.4% of runtime spent in MPI communication
2. **Gradient pyramid gather**: Single largest bottleneck at 8,074 ms per rank
3. **Limited MPI scalability**: DoG pyramid only achieves 1.33× speedup
4. **Centralized final phase**: Gathering all data to rank 0 limits scalability beyond 4 ranks

Scalability Concerns 📊

- **Strong scaling limit**: Gather operations become prohibitive as rank count increases
- **Weak scaling**: Load imbalance from rank-0-only processing increases with problem size
- **Memory pressure**: Rank 0 must hold entire gradient pyramid (high memory footprint)

Recommendations for Further Optimization

1. **Distributed descriptor computation**: Keep gradient data distributed and compute descriptors in parallel across ranks
2. **Overlap communication with computation**: Use asynchronous MPI operations where possible

3. **Optimize gather operations**: Use collective I/O or streaming approaches instead of gathering full pyramids
4. **Dynamic load balancing**: Distribute small octave processing across all ranks
5. **Memory optimization**: Avoid duplicating full gradient pyramid on rank 0

Conclusion

The hybrid MPI+OpenMP parallelization achieves a respectable **3.26× speedup** on 4 ranks with 16 threads, demonstrating effective spatial domain decomposition for the SIFT algorithm. The implementation successfully parallelizes the most computationally intensive phases (Gaussian pyramid, gradient computation, and orientation/descriptor extraction) with good efficiency.

However, the centralized gather-then-compute approach for the final phase introduces significant communication overhead (36.4% of runtime) that limits scalability. For larger-scale parallelism (8+ ranks), a fully distributed approach would be necessary to maintain efficiency and avoid the memory and communication bottlenecks associated with centralizing data on rank 0.