**Distributed Merge Sort Final Metrics Summary**

**Project Summary**

* Successfully implemented and benchmarked multiple merge sort variants using MPI and OpenMP.
* Analyzed performance across dimensions from 4 to 32K elements.

**Key Implementations**

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Parallelism Model** | **Notes** |
| serial | Sequential | Baseline |
| task0 | MPI + Sequential Merge | Scatter/Gather with root merging |
| mergepath | MPI + Sequential Merge Path | Improved merge function after gather |
| mergepath\_omp | MPI + OpenMP Merge Path | Best shared-memory speedup |
| distributed | Fully Distributed MPI | Binary tree distributed merge |

**Speedup vs Serial**

**Speed-up= Tserial​/Tparallel​**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Dimension*** | ***task0*** | ***mergepath*** | ***mergepath\_omp*** | ***distributed*** |
| *4* | ***0.01×*** | *0.10×* | *0.15×* | *0.01×* |
| *8* | ***0.09×*** | *0.28×* | *0.36×* | *0.51×* |
| *16* | ***0.72×*** | *1.00×* | *0.72×* | *1.00×* |
| *32* | ***1.35×*** | *1.50×* | *1.50×* | *2.25×* |
| *64* | ***2.95×*** | *2.50×* | *2.70×* | *4.05×* |
| *128* | ***3.33×*** | *2.46×* | *2.83×* | *2.10×* |
| *256* | ***2.91×*** | *2.29×* | *2.35×* | *3.56×* |
| *512* | ***5.15×*** | *3.29×* | *3.43×* | *5.15×* |
| *1 024* | ***5.34×*** | *5.82×* | *5.89×* | *8.84×* |
| *2 048* | ***10.17×*** | *7.30×* | *7.25×* | *9.53×* |
| *4 096* | ***12.89×*** | *7.67×* | *3.96×* | *6.98×* |
| *8 192* | ***5.43×*** | *6.14×* | *7.77×* | *8.04×* |
| *16 384* | ***9.58×*** | *8.14×* | *7.86×* | *8.23×* |
| *32 768* | ***11.70×*** | *8.09×* | *7.82×* | *11.00×* |

**insights**

* For very small problems (< 32) all parallel variants under-perform serial – MPI overhead dominates.
* task0 eventually outscales the pure serial baseline because it still scatters work, but the distributed algorithm becomes the best from 1 024 elements onward.
* mergepath\_omp is the most efficient shared-memory choice up to ≈ 4 k elements; beyond that, MPI wins.
* Large-scale (32 k) shows ~11× speed-up with both task-root and fully distributed strategies on 4 cores good .

**Main takeaway:**

* I implemented a distributed merge sort on top of MPI, debugging buffer overflows, ownership transfers, MPI pairing logic, and finally designed a fully scalable binary tree reduction pattern with dynamic resizing.
* Shared memory (mergepath\_omp) dominates for small/medium sizes.
* Distributed MPI version is fully functional & scales better at larger problem sizes >=million.
* Distributed merge initially failed on some runs due to ranks not properly synchronized between iterations.
* Memory resizing is critical in any distributed or dynamic merging algorithm.
* Send/receive sizes must always match exactly.
* In distributed algorithms, always track how many elements each rank owns after each communication round.
* unique\_ptr helps memory safety, but only if ownership transfers are carefully handled.
* MPI works once you get the message sizes, steps, and pairing rules right.