The Journal of Open Source Software

GCN10: A high-performance, MPI-parallelized framework for generating global curve number rasters

Abdullah Azzam [1] **and Huidae Cho** [1¶]

1 Department of Civil Engineering, New Mexico State University, Las Cruces, NM, USA ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 20 August 2025
Published: unpublished

Summary

The Soil Conservation Service (SCS) Curve Number (CN) method remains one of the most widely adopted approaches for estimating runoff volume and peak discharge at the watershed scale (United States Department of Agriculture, Natural Resources Conservation Service, 2019; USDA Soil Conservation Service, 1986). As the name implies, the CN value is the core input to this method, and generating high-resolution CN raster maps for large watersheds or continental domains is computationally demanding. It is becoming increasingly critical to simulate hydrologic systems at higher resolutions to achieve accurate results for different hydrologic modeling purposes, such as flood inundation mapping at the household or property scale (Wing et al., 2019). This greater level of detail improves accuracy but comes with tradeoffs in computation time, memory requirements, and the limited efficiency of many computing environments. The two major challenges are either the lack of availability of such high-resolution spatial data or the absence of efficient computing tools that can provide better processing performance than traditional GIS platforms.

The Global CN 10 m (GCN10) is a high-performance framework written in C with multi-node distributed parallelization in mind using the Message Passing Interface (MPI) (Gropp et al., 1999). It generates global CN datasets for hydrologic modeling and runoff estimation. The program uses ESA WorldCover 10 m land cover data (Zanaga et al., 2021), accessed through a GDAL virtual raster from the QGIS Curve Number Generator Plugin (Siddiqui, 2020). It combines this land cover data with the 250 m Hydrologic Soil Group (HYSOGs250m) dataset (Ross et al., 2018). CN values are assigned through a lookup table developed from (USDA NRCS, 2004b). This produces CN rasters for global or regional use. MPI enables distributed-memory parallelization using multiple processes (Gropp et al., 1999). GCN10 splits large rasters into blocks and processes them across many cores or nodes in HPC systems. The block-based design provides near-linear scaling on modern hardware (Amdahl, 1967; Gustafson, 1988). As a result, terabyte-scale datasets can be processed in hours instead of months. GCN10 is cross-platform and suitable for both research and operational hydrologic workflows. It is licensed under the GNU General Public License (GPL) v3, which is approved by the Open Source Initiative (OSI).

Statement of Need

Producing high-resolution spatial datasets for hydrology at continental or global scales is a major computational challenge. Existing approaches are limited in several ways. Desktop GIS tools are designed for small or regional tasks. They cannot process rasters that contain tens of billions of cells or manage memory needs that reach into terabytes. As a result, global CN mapping is not practical in standard computing environments.

Another problem is scalability. Many workflows are built for watershed or basin-scale studies.

They do not extend across larger domains and often require manual GIS steps. This typical workflow reduces automation and makes reproducible research difficult (Stodden et al., 2016). Handling global datasets such as ESA WorldCover at 10m resolution (Zanaga et al., 2021) and HYSOGs250m soils resampled to 10m (Ross et al., 2018) pushes these methods far beyond their design. Combining these sources into complete global rasters produces data volumes that are simply too large for serial or scripting-based workflows.

The scale of the data highlights the problem we are trying to address. In the GCN10 workflow, the globe is divided into 2,651 blocks. Each block produces 18 raster—a combination of three hydrologic conditions (poor, fair, and good), three antecedent runoff conditions (ARC I, II, and III) and two drainage conditions (drained and undrained) for dual hydrologic soil groups (USDA NRCS, 2004a). This $3 \times 3 \times 2$ combination results in 18 unique combinations for CN, and each raster has 36,000 $\times$ 36,000 cells (about 1.3 billion cells). A single raster stored as uint8 requires about 1.3 GB in memory. One block with 18 rasters requires about 23 GB. When extended across all blocks and all 18 conditions, the total data volume is more than 60 TB. Larger data types such as uint16 or float32 can double or quadruple these numbers. A single global raster contains 34.35 billion cells, and 18 such rasters are required to represent the full range of CN conditions. Running the entire globe at once would need about 7 TB of memory, even with the most compact representation. Final compressed outputs still occupy about 1.2 TB of disk space.

These requirements make it clear why serial or low-parallelism methods cannot keep up. Even scripting-based workflows with limited parallelism are too slow. Input and output (I/O) become bottlenecks, memory demands exceed what most systems provide, and runtimes stretch from days into weeks. These challenges prevent hydrologic modeling from reaching the resolution needed for accurate applications such as property-scale flood inundation mapping (Wing et al., 2019).

GCN10 addresses these barriers directly. It provides an automated workflow that uses authoritative global datasets (Ross et al., 2018; Zanaga et al., 2021). It divides the globe into blocks and processes them across many cores and nodes using MPI. The block-based design keeps memory use manageable and minimizes communication between processes. This architecture scales from individual watersheds to the entire globe. It reduces runtimes from weeks to hours and makes it possible to create CN datasets at resolutions and extents that were previously impractical.

Implementations

Google Earth Engine

Google Earth Engine (GEE) (Gorelick et al., 2017) is very effective for exploratory analysis and prototyping because it provides direct access to global datasets and a scalable cloud environment. Implementations in GEE allow rapid testing, visualization, and comparison of Curve Number generation methods without handling local storage or compute resources. However, exporting large rasters from GEE is constrained: very large files cannot be exported to Google Drive or downloaded for local use due to system limits. Thus, while GEE is excellent for interactive analysis and reproducibility,large-scale production workflows such as GCN10 require dedicated local or HPC environments to generate and store the full set of global rasters.

Python Multiprocessing

Development began with a serial Python workflow for generating CN rasters from global datasets. While functional, this approach was too slow for large-scale domains. To improve performance, a parallel version using the Python `multiprocessing.Pool` library was created (Python Software Foundation, 2023). This Python-based parallelization reduced runtime compared to the serial version but still faced major limits. Process startup and inter-process

communication are slower than in compiled languages. In addition, the Global Interpreter Lock (GIL) restricts efficient use of threads (Beazley, 2010), forcing parallelism to rely on heavy-weight process management. These factors made Python implementations inefficient and unsuitable for global CN mapping.

C MPI/OpenMP

A hybrid MPI/OpenMP parallelization scheme was implemented in C, using MPI for distributed memory parallelism and OpenMP (Dagum & Menon, 1998) for shared memory parallelism. In this design, MPI forked individual processes for each block, while the internal CN computation logic within a block was parallelized with OpenMP. The aim was to combine coarse-grain parallelism across nodes with fine-grain threading within each process. In principle this reduces MPI rank counts and exploits shared memory at the node level (Gropp et al., 1999).

In practice, the hybrid model gave little improvement. Most of the runtime was still dominated by I/O. Since GDAL raster writing is not thread-safe (GDAL Development Team, 2024), it limited the benefit of OpenMP and meant that overall throughput was similar to pure MPI. For this workload, the independence of block-level tasks made OpenMP threading unnecessary. The results of this implementation are presented in the results section.

C MPI

The workflow was then reimplemented fully in C with the Message Passing Interface (MPI) for distributed-memory parallelization (Message Passing Interface Forum, 1994). In this design, the global raster domain is partitioned into fixed spatial blocks and each block is assigned to an MPI rank for independent computation. Communication between ranks is limited to initialization and final synchronization, which minimizes message passing overhead. This structure achieves near-linear strong scaling across cores and nodes in High-Performance Computing (HPC) environments. By avoiding interpreter overhead and using efficient compiled code, the C MPI implementation delivers order-of-magnitude speedups compared to the Python versions.

The overall design of the MPI workflow is shown in Figure 1. The flowchart illustrates how blocks are distributed across ranks, processed independently, and synchronized at the end of execution.

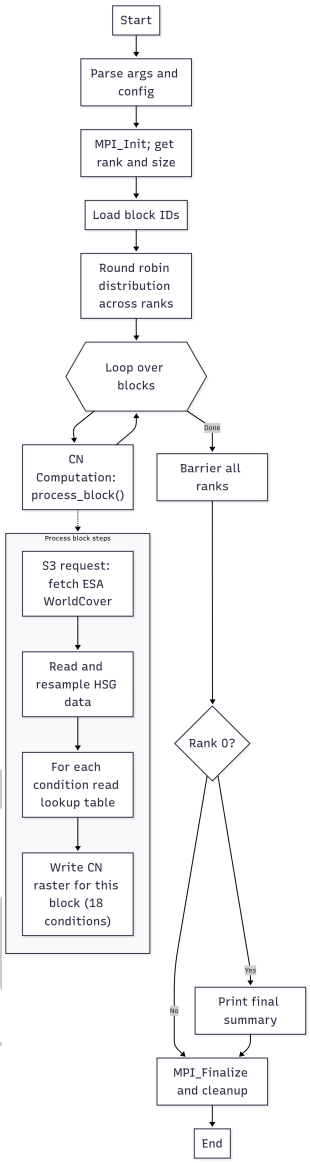

Figure 1: Workflow of GCN10 MPI processing and CN computation logic.

Testing Environment and System Architecture

A consistent protocol was applied across all three implementations. The design included the following elements:

- Hardware and data
  - Benchmarks ran on the system described in Table @ref{tab:hardware}.
  - Test region: 32 spatial blocks covering Utah, Nevada, Colorado, New Mexico, Arizona, and the entire state of Texas.
  - All blocks contained valid data.
  - Inputs: ESA WorldCover (10m) (Zanaga et al., 2021) and HYSOGs250m soils resampled to 10m (Ross et al., 2018).
  - Each block was processed for all 18 CN conditions.
- I/O and storage
  - Local NVMe SSD storage only; no network file systems used.

  - Outputs written as GeoTIFF rasters (Ritter & Ruth, 1994) with LZW compression (Welch, 1984), uint8 pixels.
  - Internal tiling enabled with GDAL default tile sizes (GDAL Development Team, 2020).
  - GDAL cache left at default settings.
  - Progress messages and debug logs disabled to avoid I/O noise.
- Timing and validation
  - Runtime measured with POSIX time (user, system, elapsed wall time) (*POSIX.1-2024*, 2024).
  - Validation by checking output file counts and inspecting rasters with `gdalinfo` (GDAL Development Team, 2020).
  - No failures observed for the 32-block test set.
- Python multiprocessing.Pool
  - Pool size: 16 workers.
  - One warmup run, followed by one measured run.
  - Runtime was long relative to other implementations, so replicates were not pursued.
- Hybrid MPI/OpenMP
  - MPI ranks launched to fill the node.
  - OMP_NUM_THREADS varied across runs for tuning.
  - Each configuration executed 30 times to obtain stable runtime statistics.
- MPI standalone
  - MPI ranks launched to fill the node for each scenario.
    * Options: `--bind-to hwthread --report-bindings`
    * Reason: Each rank was pinned to a unique logical CPU to avoid OS scheduling migration and ensure consistent cache locality. Binding at the hardware-thread level provided reproducible performance measurements. The binding report was enabled to verify correct placement.
  - Each configuration executed 30 times to obtain stable runtime statistics.

All implementations were benchmarked on a single workstation with the specifications in Table @ref{tab:hardware}.

Table 1: Hardware and system specifications for performance testing. {#tab:hardware}

Component	Specification
Architecture	x86_64 (64-bit)
CPU Model	Intel Core i9-13900KS (13th Gen)
Memory	125 GiB
Physical cores	24
Total threads	32
Max frequency	6.0 GHz
NUMA nodes	1
Operating system	Linux 6.6.23
Compiler collection	GNU Compiler Collection (GCC) 13.2.0
MPI Compiler	Open MPI version 4.1.4

Results

Python Multiprocessing

The Python `multiprocessing.Pool` implementation with 16 workers, required 21,207 seconds (~5 hours 54 minutes) to process the 32-blocks test set. This experiment confirmed that Python overheads in memory handling and process management make it unsuitable for global-scale CN

mapping. Assuming ideal speedup, running the same tests as the C implementations would have taken ~230 days.

C MPI/OpenMP

The hybrid MPI/OpenMP implementation was tested by varying both the number of MPI ranks (N) and OpenMP threads (T). Increasing the number of threads within a rank did not reduce runtime significantly (Figure 2). Speedup scaled primarily with the number of MPI ranks, while threading overhead and GDAL's lack of thread-safe raster writing limited the benefits of OpenMP parallelism. Allocating cores to OpenMP threads effectively reduced the number of available MPI ranks, underutilizing hardware that could otherwise process additional blocks in parallel.

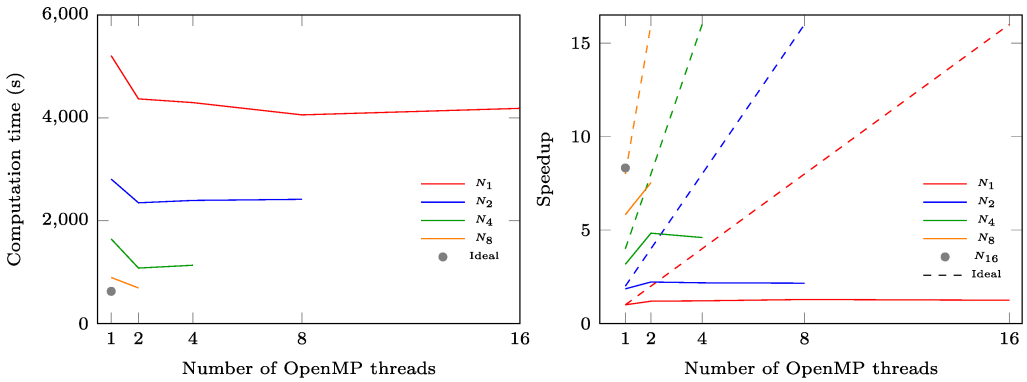

Figure 2: Computation time and speedup for hybrid MPI/OpenMP across threads and ranks.

C MPI

The standalone MPI implementation achieved strong scaling across ranks, with computation times decreasing from ~7,800 seconds at 1 rank to ~845 seconds at 16 ranks (Figure 3). This performance confirms that the implementation can process increasingly large workloads in shorter wall times as ranks increase.

@ref{tab:mpi-summary} summarizes performance metrics for the MPI-only implementation up to 16 ranks, including runtime, CPU usage, speedup, and efficiency.

Table 2: Summary of MPI-only performance metrics across different rank counts.{#tab:mpi-summary}

Ranks	Mean runtime (s)	Mean CPU usage (%)	CPU usage per rank (%)	Speedup	Efficiency
1	7865.89	72.27	72.27	1.00	1.00
2	4633.28	127.01	63.50	1.69	0.84
4	2212.46	263.88	65.97	3.55	0.88
8	1186.80	588.34	73.54	6.62	0.82
16	845.12	1196.94	74.80	9.30	0.58

The strong scaling behavior in terms of runtime is observed. Each doubling of ranks yielded substantial reductions in wall time, with diminishing returns at higher counts because of parallel overhead (Figure 3).

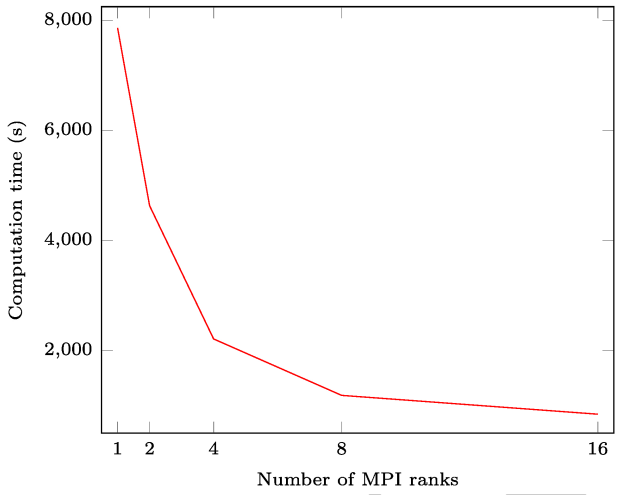

Figure 3: Computation time of MPI implementation across ranks.

Derived performance metrics including speedup, efficiency, and CPU utilization are also cacluated (Figure 4). Speedup increased nearly linearly with rank count, while efficiency remained close to ideal through 8 ranks and declined slightly at 16 ranks. Total CPU usage rose smoothly with ranks, and per-rank usage stayed stable, indicating balanced utilization without oversubscription.

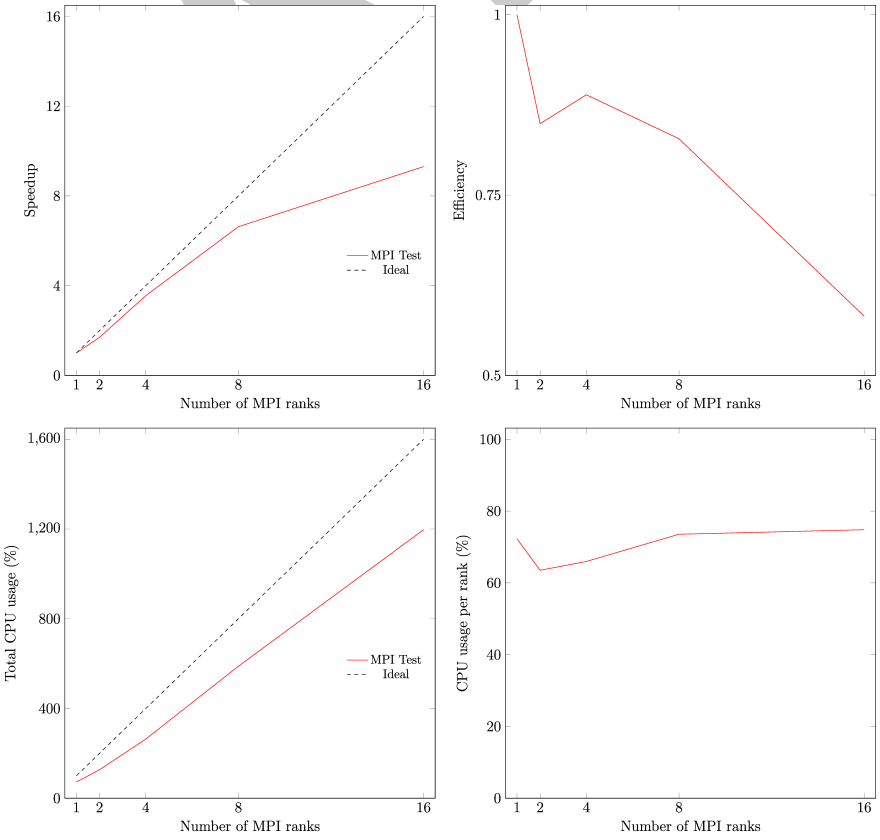

Figure 4: MPI performance metrics across ranks, showing speedup, efficiency, and CPU utilization.

Acknowledgments

We acknowledge the New Mexico Water Resources Research Institute (NM WRRI) and the New Mexico State Legislature for their support and resources, which were essential for this project. This work is funded by the NM WRRI and the New Mexico State Legislature under the grant number NMWRRI-SG-FALL2024.

References

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. *AFIPS Conference Proceedings*, *30*, 483–485.

Beazley, D. (2010). Understanding the python GIL. *Python Conference (PyCON)*.

Dagum, L., & Menon, R. (1998). OpenMP: An industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, *5*(1), 46–55. https://doi.org/10.1109/99.660313

GDAL Development Team. (2020). *GDAL/OGR geospatial data abstraction software library*.

GDAL Development Team. (2024). *RFC 101: Raster dataset read-only thread-safety*. https://gdal.org/en/stable/development/rfc/rfc101_raster_dataset_threadsafety.html

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., & Moore, R. (2017). Google earth engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, *202*, 18–27. https://doi.org/10.1016/j.rse.2017.06.031

Gropp, W., Lusk, E., & Skjellum, A. (1999). *Using MPI: Portable parallel programming with the message passing interface*. MIT Press.

Gustafson, J. L. (1988). Reevaluating amdahl's law. *Communications of the ACM*, *31*(5), 532–533.

IEEE/Open Group Standard for Information Technology–Portable Operating System Interface (POSIX™) Base Specifications, Issue 8 (IEEE Std 1003.1-2024). (2024). IEEE; The Open Group. https://doi.org/10.1109/IEEESTD.2024.10555529

Message Passing Interface Forum. (1994). *MPI: A message-passing interface standard*. University of Tennessee.

Python Software Foundation. (2023). *Python multiprocessing library documentation*.

Ritter, N., & Ruth, M. (1994). *GeoTIFF format specification*. SPOT Image.

Ross, C. W., Prihodko, L., Anchang, J., Kumar, S., Ji, W., & Hanan, N. P. (2018). HYSOGs250m, global gridded hydrologic soil groups for curve-number-based runoff modeling. *Scientific Data*, *5*, 180091. https://doi.org/10.1038/sdata.2018.91

Siddiqui, A. R. (2020). *Curve number generator: A QGIS plugin to generate curve number layer from land use and soil*. https://github.com/ar-siddiqui/curve_number_generator.

Stodden, V., Seiler, J., & Ma, Z. (2016). Enhancing reproducibility for computational methods. *Science*, *354*(6317), 1240–1241. https://doi.org/10.1126/science.aah6168

United States Department of Agriculture, Natural Resources Conservation Service. (2019). *National engineering handbook, part 630 hydrology* ((210–630–H., Amend. 88)). https://directives.nrcs.usda.gov/sites/default/files2/1712930592/29495.pdf

USDA NRCS. (2004a, July). *National engineering handbook, part 630 hydrology: Chapter 7 - hydrologic soil group (210-VI-NEH)*. USDA eDirectives. hhttps://directives.nrcs.usda.gov/sites/default/files2/1712930597/11905.pdf

USDA NRCS. (2004b, July). *National engineering handbook, part 630 hydrology: Chapter 9 - hydrologic soil-cover complexes (210-VI-NEH)*. USDA eDirectives. https://directives.sc.egov.usda.gov/landingpage/79824ad0-6672-4e3b-b100-86d87993f172

USDA Soil Conservation Service. (1986). *Urban hydrology for small watersheds, TR-55* (Technical Release 55). U.S. Department of Agriculture.

Welch, T. (1984). A technique for high-performance data compression. *Computer*, *17*, 8–19.

Wing, O. E. J., Bates, P. D., Sampson, C. C., Smith, A. M., Johnson, K. A., & Erickson, T. A. (2019). New insights into flood risk from a global-scale catastrophe model. *Water Resources Research*, *55*(3), 185–199. https://doi.org/10.1029/2018WR024205

Zanaga, D., Van De Kerchove, R., De Keersmaecker, W., Souverijns, N., Brockmann, C., Kirches, G., Wevers, J., Cartus, O., Santoro, M., & Fritz, S. (2021). ESA WorldCover 10 m 2020 product description and validation report. *Zenodo*. https://doi.org/10.5281/zenodo.5571936