Accelerating Large-scale, Hi-res Raster Data Processing Workflows

MING-CHIEH HU, Delft University of Technology, The Netherlands

1 Introduction

At Readar, high-resolution aerial images are used to compute large-scale photovoltaic (PV) potential statistics, providing deeper insights into buildings. This process relies on tiled raster data with a resolution of 8cm covering the entire Netherlands.

However, handling such high-resolution data becomes the main bottleneck for subsequent processing steps. Even with parallel computation and optimized file I/O, the workflow remains slow. The existing implementation uses Python, which is not inherently slower than C or C++ for many operations because libraries like NumPy internally use highly optimized C or C++ code accessible through Python APIs.

In this study, we aim to address this bottleneck by exploring and comparing different methods through a series of experiments. Specifically, we investigate and evaluate three approaches: the **Vector–Raster method** (baseline solution), the **Raster–Raster method**, and a **Hybrid method**, the latter two being newly proposed in this work.

The goal is to systematically analyze their performance and suitability for the task at hand, considering both accuracy and computational efficiency. The scope of this study is limited to:

- Implementations in Python, using libraries such as NumPy and Rasterio;
- Excluding GPU-based acceleration, to ensure the methods remain accessible and portable across different computing environments and hardware configurations.

By comparing these methods under controlled conditions, this work aims to identify practical solutions that balance memory and time efficiency.

The structure of this paper is as follows. Section 3 describes the sample data used to evaluate the proposed approaches. Section 2 outlines the theoretical background and methodology. Section 4 presents the experimental results and analysis. Finally, Section 5 concludes the study and discusses potential directions for future work.

2 Methodology

This task involves calculating and generating statistics on the coverage of PV panel installations on building rooftops, primarily by measuring area. It uses both raster and vector data and is part of the routine raster processing workflow. Similar to aggregation functions, this task focuses on analysis rather than geometry. The key focus is PV panels located on building rooftops—think of it like a cookie cutter: classified rasters are "cut" using numerous building polygons to isolate and analyze only the relevant areas.

2.1 Vector-Raster Method (Baseline)

As shown in Algorithm 1, with a given raster file, the method begins by querying a PostGIS database to retrieve all building geometries contained within the file's spatial bounds. Then, for each building polygon, it performs a series of operations in parallel: the raster is opened and clipped to the extent of the polygon geometry, after which the resulting masked image is converted to a binary map where pixels equal to 3 become 1 and all others become 0. The pixels with value 1, representing photovoltaic areas, are counted, and the building's identifier, PV pixel count, and total pixel count are recorded. The results from all buildings are then compiled into a DataFrame for further analysis or storage.

2.2 Raster-Raster Method (Proposed)

The Vector-Raster (baseline) method is simple and performs well with a small number of building polygons. However, beyond a certain threshold, performance degrades significantly because each polygon requires a separate raster file I/O operation.

Author's address: Ming-Chieh Hu, m.hu-5@student.tudelft.nl, Delft University of Technology, Delft, The Netherlands.

Algorithm 1 Baseline PV Detection

```
Require: Raster $\mathcal{R}$, polygon layer $P = \{p_1, p_2, \ldots, p_n\}$
Ensure: DataFrame $\mathcal{D}$ with pixel statistics per polygon
for all $p_i \in P$ do                                          ▷ This is processed in parallel
    $data \leftarrow$ read_raster($\mathcal{R}$)
    $intersection \leftarrow$ clip($data, p_i$)
    $intersection \leftarrow (intersection == 3)?1 : 0$
    $counts \leftarrow intersection.sum()$
    Record ($\text{id}_{p_i}, counts, intersection.size$) into $\mathcal{D}$
end for
return $\mathcal{D}$
```

When dealing with a very large number of polygons, it is generally more efficient to convert the vector data into raster format. This approach benefits from a technique known as, ironically, vectorization. In NumPy, vectorization refers to performing operations on entire arrays without explicit Python loops. By replacing iterative processes with vectorized operations, we can take advantage of NumPy's underlying C implementation for faster and more efficient computation. In other words, counting pixel values for many small patches in a Python loop is much slower than processing one large raster all at once.

The task is transformed from a vector–raster intersection to a raster–raster multiplication, replacing explicit for-loops with a raster-based data structure. Rasterization is therefore key to our approach. We propose a rasterized solution utilizing `rasterio.mask.raster_geometry_mask()`, manually create an integer map, and read the raster file in tiles, performing raster multiplication tile by tile

This method is designed with memory efficiency in mind as the raster data is read and processed in tiles. The general framework is illustrated in Algorithm 2.

Algorithm 2 Rasterized PV Detection

```
Require: Raster $\mathcal{R}$, polygon layer $P = \{p_1, p_2, \ldots, p_n\}$
Ensure: DataFrame $\mathcal{D}$ with pixel statistics per polygon
for all $p_i$ in $P$ do
    $map \leftarrow$ rasterize_polygon($p_i$)
    $size \leftarrow$ np.count_nonzero($map$)
    Record ($id, size$) into $\mathcal{D}$
    Record $map$ into $maps$                                  ▷ Burn polygon IDs into $maps$
end for
$tiles \leftarrow$ divide_into_subtiles($maps$)
for all $tile$ in $tiles$ do
    $data \leftarrow$ read_raster($\mathcal{R}, tile.bounds$)
    $intersection \leftarrow tile \odot data$
    $counts \leftarrow$ np.bincount($intersection$)
end for
for all $id \neq 0$ do
    Record ($counts[id]$) into $\mathcal{D}$
end for
return $\mathcal{D}$
```

2.3 Hybrid Method (Proposed, best-performing)

The baseline Vector-Raster method demonstrates optimal performance when processing a limited number of building polygons, primarily due to reduced I/O operations. Conversely, the Raster-Raster method excels when handling large polygon datasets with extensive spatial coverage. As illustrated in Section 3, building distribution across the study area is highly heterogeneous, with covered areas varying significantly within individual tiles. A single tile may contain completely empty regions—ideally suited for the Vector-Raster method—alongside densely built areas where the Raster-Raster method performs optimally.

2.3.1 *Adaptive Method Selection*

A straightforward approach would involve establishing a polygon count threshold to determine method selection. However, given the substantial variation in tile sizes shown in Figure 2, simple polygon count thresholds prove unreliable. A more robust solution employs **polygon occupancy rate** as the decision metric:

$$OccupancyRate = \frac{\text{Area(polygons)}}{\text{Area(raster)}} \times 100\% \tag{1}$$

For each sub-tile, the processing strategy follows:

$$\text{Do} \begin{cases} \textbf{Nothing}, & \text{if } \textit{no building polygon found} \\ \textbf{Vector-Raster method}, & \text{if } OccupancyRate < threshold\ \% \\ \textbf{Raster-Raster method}, & \text{if } OccupancyRate \geq threshold\ \% \end{cases} \tag{2}$$

2.3.2 *Spatial Partitioning (Tiling)*

To maximize the effectiveness of this adaptive approach, we implement hierarchical spatial partitioning by subdividing tiles into sub-tiles. This finer granularity also serves a critical purpose, as it creates regions that approach the extreme cases where each base method excels—either sparsely populated areas favoring Vector-Raster processing or densely built zones optimal for Raster-Raster operations.

This adaptive partitioning strategy enables the algorithm to dynamically select the most efficient processing method based on local polygon density characteristics rather than global tile properties, resulting in significant performance improvements across diverse urban morphologies.

3 Dataset

The dataset used in this study is designed to measure PV potential; accordingly, the tile files follow the naming format `pv_xx_yy.tif` (shown here in monospaced font). Throughout this report, this format will be used, or simply abbreviated as tile `xx_yy`.

Four sample tiles were selected to test various scenarios. Their overall statistics are presented in Table 1, and visualizations are provided in Figures 1 and 2. Note that while the pixels themselves are square, the tiles vary slightly in shape and size.

Each tile typically covers an area of around one hundred million square meters, with dimensions exceeding $100,000 \times 100,000$ pixels. Additionally, the distribution of buildings can differ from tile to tile; this variation is addressed using the *OccupancyRate* metric defined in Eq. 1.

Tile	Buildings No.	Building Area (m^2)	Tile Area (m^2)	Occupancy Rate
`pv_18_42.tif`	102,584	9,422,577.69	99,656,663.04	9.46%
`pv_18_43.tif`	54,215	5,103,095.15	102,770,933.76	4.97%
`pv_20_50.tif`	60,522	6,309,149.57	99,656,663.04	6.33%
`pv_24_52.tif`	13,008	1,879,524.32	99,656,663.04	1.89%

Table 1. Building coverage statistics for selected tiles

4 Experiments

In Section 2, three approaches are explained: the Vector–Raster Method (baseline solution), the Raster–Raster Method, and a Hybrid Method.

To conduct a fair comparative evaluation, all methods must be assessed under consistent experimental conditions. While the Vector–Raster Method (baseline) can operate effectively without spatial partitioning, the Raster–Raster Method requires substantial memory resources that exceed available system capacity when processing entire tiles simultaneously. To enable fair comparison, all methods are evaluated using the same tiling approach, which also

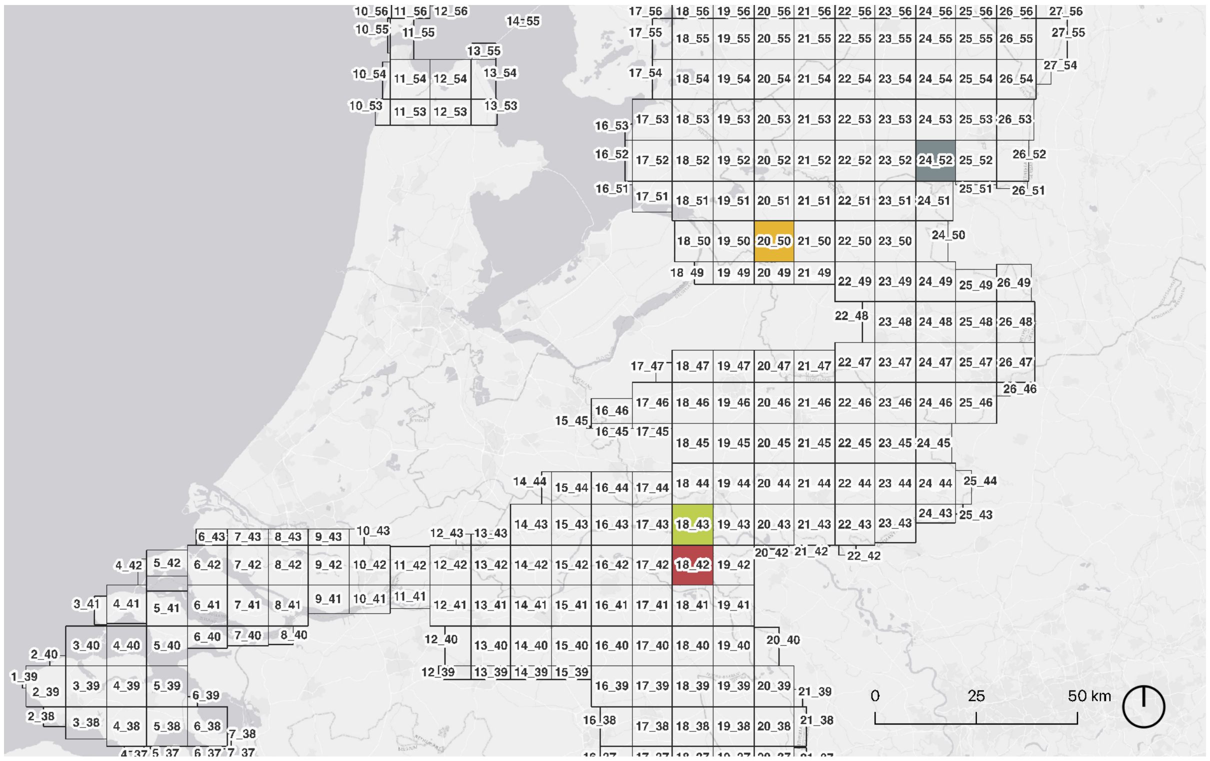
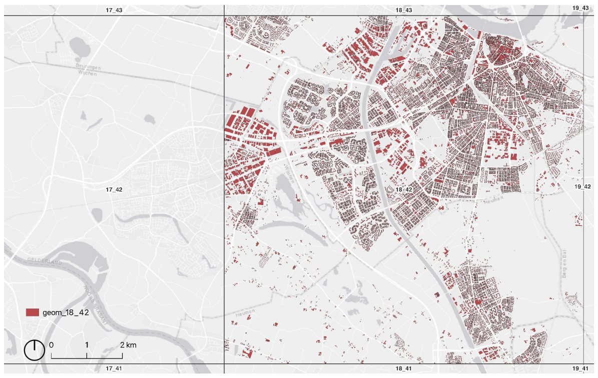
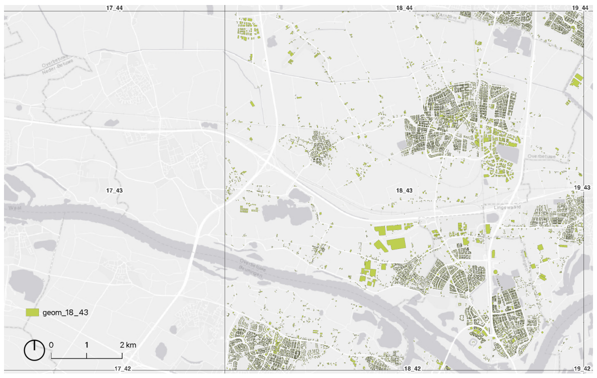


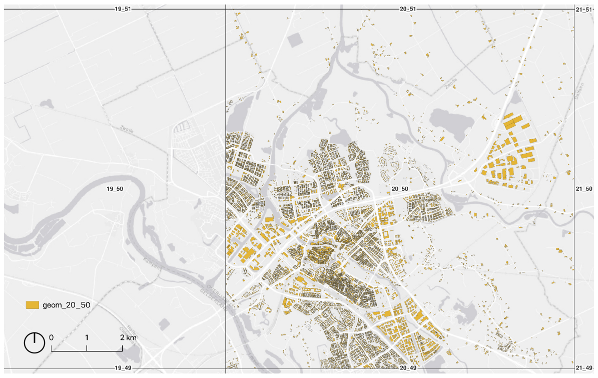
Fig. 1. Tile division overview



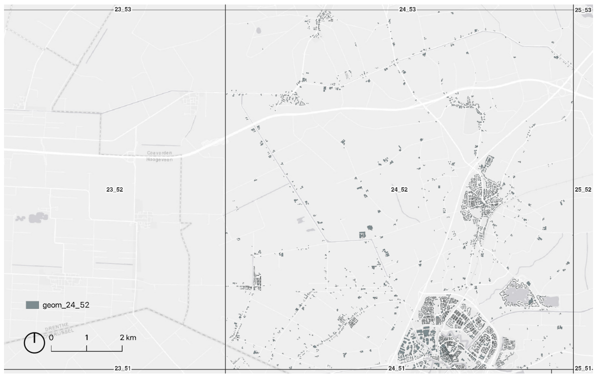
(a) Building polygons in tile 18_42



(b) Building polygons in tile 18_43



(c) Building polygons in tile 20_50



(d) Building polygons in tile 24_52

Fig. 2. Building footprints in the extent of the 4 tiles

allows the Hybrid Method to achieve performance gains through adaptive method selection based on polygon occupancy rate within sub-tiles.

To isolate the algorithmic contributions from tiling effects, the experimental evaluation consists of two components:

(1) **Performance evaluation across tiling schemes**: Evaluate all three methods (Vector-Raster, Raster-Raster, and Hybrid) across a range of tiling schemes from 1×1 to 32×32 sub-tiles. This analysis determines each method's performance characteristics across different spatial granularities and identifies optimal tiling configurations for each method.
(2) **Parameter optimization**: Evaluate and determine the optimal *OccupancyRate* threshold for the Hybrid Method.

This experimental design ensures that performance comparisons reflect genuine algorithmic differences rather than memory limitations or implementation-specific optimizations. The results and analysis are presented in Sections 4.1, 4.2. The hardware and software specification for experiments is listed in Table 7.

4.1 Comprehensive Evaluation Across Different Tiling Schemes

Currently, we have several variables to consider: the *OccupancyRate* threshold, the tiling scheme, and the choice of method. To enable fair comparisons, we made a reasonable assumption: the *OccupancyRate* threshold should lie between 1.89% (from tile `24_52`) and 9.46% (from tile `18_42`), based on their observed overall occupancy rates. After examining Table 1, we selected a threshold of 5%, which will be further evaluated in Section 4.2.

The comprehensive evaluation tests all three methods (Vector-Raster, Raster-Raster, and Hybrid) across a range of tiling schemes, from 1×1 to 32×32 sub-tiles. The corresponding approximate tile sizes are 120000×120000, 60000×60000, 30000×30000, 15000×15000, 7500×7500, and 3800×3800. Note that the exact tile sizes vary slightly.

As shown in Table 2, the proposed Hybrid method performs significantly better than the other methods on tile `18_42`, while delivering comparable or slightly improved results on tiles `18_43`, `20_50`, and `24_52`. Here, "OOM" indicates an out-of-memory error. Additional analysis plots are provided in Appendix A.2.

What is particularly interesting is how the optimal tiling scheme varies across different tiles... for tiles `18_42` and `18_43`, the sweet spot is around 16×16 subdivision, while tile `20_50` performs best with 8×8 tiling. This suggests that the optimal granularity depends on the specific data characteristics and occupancy patterns.

Interestingly, chunking large rasters into smaller tiles yields substantial runtime improvements—even greater than the gains from combining the two methods into a hybrid approach. For instance, the Vector-Raster baseline method takes 985 seconds without tiling, but only 216 seconds with 8×8 tiling. This trend holds across most files. The improvement is due not only to the higher chance of skipping sparse tiles at finer granularity but also to the non-linear reduction in the time complexity of parallel task allocation. For the Raster-Raster method, the runtime also improves since the reduction in file I/O time outweighs the added loop overhead. However, there is clearly a point of diminishing returns... pushing to 32×32 subdivision often degrades performance again, likely because the overhead of managing too many small tiles starts to outweigh the computational benefits.

Tile `24_52` consistently shows the fastest processing (around 2-4ms per million pixels for most configurations), while tile `18_42` requires significantly more computation time (up to 63ms per million pixels without tiling). This dramatic difference aligns with their occupancy rates and suggests that sparse tiles with lower occupancy are inherently easier to process, regardless of the method used.

As expected, the Vector-Raster method performs better when the *OccupancyRate* is low, whereas the Raster-Raster method is more efficient at higher occupancy levels. The Hybrid method consistently outperforms both by adapting to the characteristics of the data and leveraging the strengths of each approach. For our use case file size, it is found that the 16×16 tile division is the best performing one, and that roughly equals to 7500×7500 in tiling pixel size. The hybrid approach never falls below the performance of the better base method, which validates our assumption that intelligent method selection based on local occupancy can indeed provide consistent improvements across diverse data characteristics.

Tile	Method	Runtime	1×1	2×2	4×4	8×8	16×16	32×32
`pv_18_42.tif`	Vector-Raster	Total (s)	985.505	398.752	245.695	216.634	253.420	323.891
		Per Million Pixels (ms)	63.290	25.608	15.779	13.912	16.275	20.800
	Raster-Raster	Total (s)	OOM	337.848	296.932	227.194	186.532	188.549
		Per Million Pixels (ms)	OOM	21.697	19.069	14.590	11.979	12.109
	Hybrid	Total (s)	OOM	238.201	217.938	148.166	**134.384**	154.702
		Per Million Pixels (ms)	OOM	15.297	13.996	9.515	**8.630**	9.935
		Total Pixels						15,571,353,600
`pv_18_43.tif`	Vector-Raster	Total (s)	281.743	147.008	115.622	115.935	134.343	180.174
		Per Million Pixels (ms)	17.545	9.155	7.200	7.220	8.366	11.220
	Raster-Raster	Total (s)	OOM	298.452	265.435	190.590	171.499	134.049
		Per Million Pixels (ms)	OOM	18.586	16.530	11.869	10.680	8.348
	Hybrid	Total (s)	281.743	236.581	167.613	103.805	**95.701**	101.246
		Per Million Pixels (ms)	17.545	14.733	10.438	6.464	**5.960**	6.305
		Total Pixels						16,057,958,400
`pv_20_50.tif`	Vector-Raster	Total (s)	473.175	255.222	156.194	137.599	182.913	210.498
		Per Million Pixels (ms)	30.388	16.390	10.031	8.837	11.747	13.518
	Raster-Raster	Total (s)	OOM	321.916	282.401	213.208	161.890	130.942
		Per Million Pixels (ms)	OOM	20.674	18.136	13.692	10.397	8.409
	Hybrid	Total (s)	OOM	140.365	165.669	**101.345**	**101.555**	112.995
		Per Million Pixels (ms)	OOM	9.014	10.639	**6.508**	**6.522**	7.257
		Total Pixels						15,571,353,600
`pv_24_52.tif`	Vector-Raster	Total (s)	35.026	34.996	33.421	37.349	50.348	66.506
		Per Million Pixels (ms)	2.249	2.247	2.146	2.399	3.233	4.271
	Raster-Raster	Total (s)	OOM	276.111	238.266	186.414	125.425	82.681
		Per Million Pixels (ms)	OOM	17.732	15.302	11.972	8.055	5.310
	Hybrid	Total (s)	35.026	**31.998**	38.161	40.804	44.051	47.890
		Per Million Pixels (ms)	2.249	**2.055**	2.451	2.620	2.829	3.076
		Total Pixels						15,571,353,600

Table 2. LONG

4.2 Parameter Optimization Across Different *OccupancyRate* Thresholds

As the effect of tiling scheme is displayed in Section 4.1, the best-performing parameter to divide tiles is fixed to be 16×16. In this part, we further evaluate the Hybrid method using candidate *OccupancyRate* thresholds of 0.02, 0.03, 0.04, 0.05, 0.06, 0.07 and 0.08.

From the Tables 3, 4, 5 and 6 one can see that this experiment can not really find the best best param out, as the optimal threshold varies depending on the spatial distribution of buildings within each tile. Notably, only tile `18_42` demonstrates improved performance when utilizing a higher proportion of the Raster-Raster method, while the other tiles perform better with greater reliance on the Baseline method. tile `18_42` achieves minimum runtime at 0.06 threshold, while tiles `18_43` and `20_50` perform best at 0.05 and 0.08 respectively, and tile `24_52` shows optimal performance at 0.07.

This variation reflects the adaptive nature of the hybrid approach, where tiles with different building densities and spatial patterns benefit from different decision boundaries between the Baseline and Raster-Raster Methods. However, the performance differences between threshold values are relatively small, with runtime variations typically within 10-15% of the optimal value. Figures 7, 8, 9 and 10 showcase the detailed plots of the proportion of each case.

Metric \ Threshold	**0.02**	**0.03**	**0.04**	**0.05**	**0.06**	**0.07**	**0.08**
Tiles Processed with Baseline (%)	25.8	34.0	39.1	41.0	43.4	44.5	46.9
Tiles Processed with Solution 1 (%)	66.8	58.6	53.5	51.6	49.2	48.0	45.7
Tiles Skipped (%)	7.4	7.4	7.4	7.4	7.4	7.4	7.4
Runtime: Baseline (s)	10.900	18.525	22.892	25.616	28.721	30.442	35.268
Runtime: Solution 1 (s)	125.441	123.893	109.959	105.627	100.357	102.071	94.030
Runtime: Total (s)	147.787	154.332	144.599	142.793	**140.426**	144.001	**140.517**
Runtime Per Million Pixels (ms)	9.491	9.911	9.286	9.170	**9.018**	9.248	**9.024**
Total Pixels							15,571,353,600

Table 3. *OccupancyRate* threshold analysis of tile 18_42

Metric \ Threshold	**0.02**	**0.03**	**0.04**	**0.05**	**0.06**	**0.07**	**0.08**
Tiles Processed with Baseline (%)	42.2	50.0	53.1	56.6	61.3	64.1	65.6
Tiles Processed with Solution 1 (%)	47.7	39.8	36.7	33.2	28.5	25.8	24.2
Tiles Skipped (%)	10.2	10.2	10.2	10.2	10.2	10.2	10.2
Runtime: Baseline (s)	16.556	22.019	24.427	28.531	36.547	39.957	45.169
Runtime: Solution 1 (s)	92.207	79.998	71.493	66.652	62.438	55.585	53.913
Runtime: Total (s)	117.400	110.513	104.184	**103.451**	107.556	**104.080**	107.558
Runtime Per Million Pixels (ms)	7.311	6.882	6.488	**6.442**	6.698	**6.482**	6.698
Total Pixels							16,057,958,400

Table 4. *OccupancyRate* threshold analysis of tile 18_43

Metric \ Threshold	**0.02**	**0.03**	**0.04**	**0.05**	**0.06**	**0.07**	**0.08**
Tiles Processed with Baseline (%)	44.5	53.5	57.4	58.2	59.8	61.7	62.5
Tiles Processed with Solution 1 (%)	48.0	39.1	35.2	34.4	32.8	30.9	30.1
Tiles Skipped (%)	7.4	7.4	7.4	7.4	7.4	7.4	7.4
Runtime: Baseline (s)	14.721	21.941	25.225	26.157	28.112	28.966	30.264
Runtime: Solution 1 (s)	90.758	75.132	68.639	68.231	65.156	59.117	57.496
Runtime: Total (s)	114.718	106.420	103.111	103.446	102.286	**96.924**	**96.658**
Runtime Per Million Pixels (ms)	7.367	6.834	6.622	6.643	6.569	**6.225**	**6.207**
Total Pixels							15,571,353,600

Table 5. *OccupancyRate* threshold analysis of tile 20_50

Metric \ Threshold	**0.02**	**0.03**	**0.04**	**0.05**	**0.06**	**0.07**	**0.08**
Tiles Processed with Baseline (%)	56.6	64.1	66.8	68.4	71.5	71.5	72.7
Tiles Processed with Solution 1 (%)	21.5	14.1	11.3	9.8	6.6	6.6	5.5
Tiles Skipped (%)	21.9	21.9	21.9	21.9	21.9	21.9	21.9
Runtime: Baseline (s)	15.640	20.585	22.558	23.794	27.664	27.783	30.250
Runtime: Solution 1 (s)	34.660	23.469	19.253	16.795	11.896	11.947	9.865
Runtime: Total (s)	53.563	47.342	45.165	43.913	**43.234**	**42.939**	43.594
Runtime Per Million Pixels (ms)	3.440	3.040	2.901	2.820	**2.777**	**2.758**	2.800
Total Pixels							15,571,353,600

Table 6. *OccupancyRate* threshold analysis of tile 24_52

5 Conclusion

This study presented a comprehensive evaluation of three computational approaches for raster-vector processing: the Vector-Raster method (baseline), the Raster-Raster method, and a Hybrid method that adaptively selects between the two based on local polygon occupancy rates.

5.1 Key Findings

Our initial hypothesis that the Hybrid method would outperform individual approaches through adaptive method selection was confirmed. The Hybrid approach consistently matched or exceeded the performance of the better base method across all test cases, validating our assumption that intelligent method selection based on local occupancy characteristics can provide consistent improvements across diverse spatial data patterns.

Through exhaustive testing across multiple tiling schemes (from 1×1 to 32×32 subdivisions) and occupancy rate thresholds (0.02 to 0.08), we demonstrated that the optimal configuration varies significantly with spatial data characteristics. The Hybrid method achieved performance gains of up to 733% compared to the baseline method on dense tiles like 18_42, while maintaining comparable efficiency on sparse tiles.

However, the most significant and unexpected finding was that spatial tiling itself provides substantial performance improvements that often exceed the gains from algorithmic hybridization. For instance, the Vector-Raster baseline method improved from 985 seconds without tiling to 216 seconds with 8×8 tiling on tile 18_42—a 4.5× speedup. This improvement stems from both the ability to skip sparse regions and the non-linear reduction in computational complexity through parallel task allocation.

5.2 Limitations

Several limitations should be acknowledged. First, our evaluation was conducted on a limited set of four representative tiles, which may not capture the full spectrum of spatial data characteristics encountered in operational environments. Second, the memory constraints that necessitated tiling for the Raster-Raster method may have influenced the comparative results, potentially masking the true performance characteristics of each algorithm when unconstrained by memory limitations. Third, the occupancy rate threshold optimization revealed that optimal parameters are highly data-dependent, suggesting that a fixed threshold may not be suitable for heterogeneous datasets requiring automated processing pipelines.

5.3 Future Work

Several research directions emerge from this work. First, developing adaptive threshold selection mechanisms that can automatically determine optimal occupancy rate parameters based on real-time analysis of spatial data characteristics would enhance the practical applicability of the Hybrid method. Second, investigating the scalability of these approaches to larger datasets and different spatial resolutions would provide insights into their operational viability. Third, exploring alternative spatial partitioning strategies beyond regular grid tiling—such as adaptive quadtree decomposition or feature-aware partitioning—could potentially yield further performance improvements. Finally, extending the evaluation to include additional computational methods and diverse geospatial processing tasks would help establish the broader applicability of hybrid computational approaches in geomatics applications.

The demonstrated effectiveness of spatial tiling suggests that future research should prioritize spatial data structures and partitioning strategies as fundamental components of high-performance geospatial computing, rather than treating them merely as implementation details.

A Additional Information

A.1 Hardware and Software Specification

Component	Specification
Device	MacBook Pro, 14-inch (2021)
Chip	Apple M1 Pro
CPU	8-core (6 performance + 2 efficiency)
Memory Bandwidth	200 GB/s
Memory	16 GB unified memory
Operating System	macOS Sonoma 14.7.1
Python Version	3.13.0

Table 7. Hardware and software specification

A.2 Comprehensive Evaluation: Detailed Plots

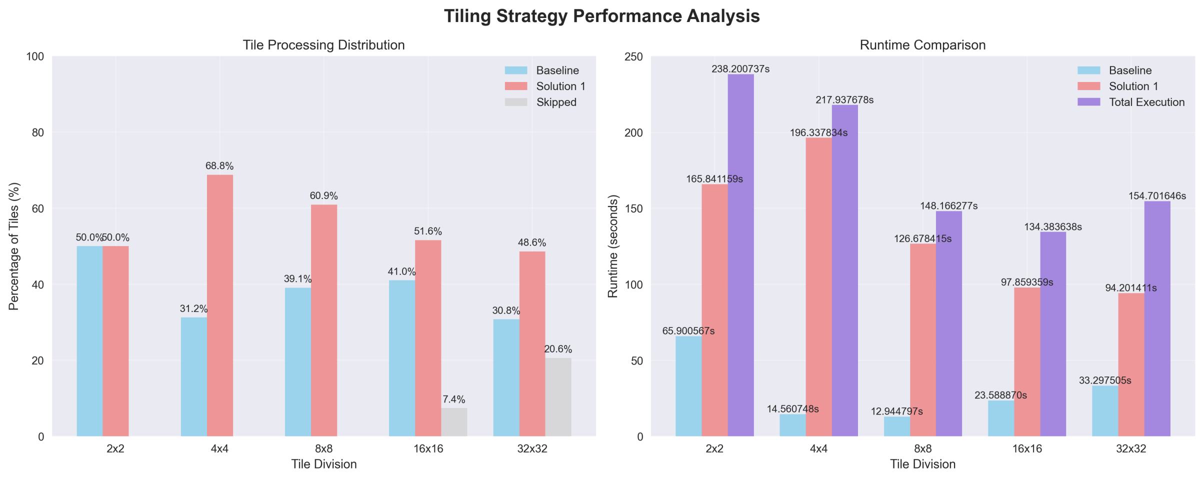


Fig. 3. Tile division analysis of tile 18_42

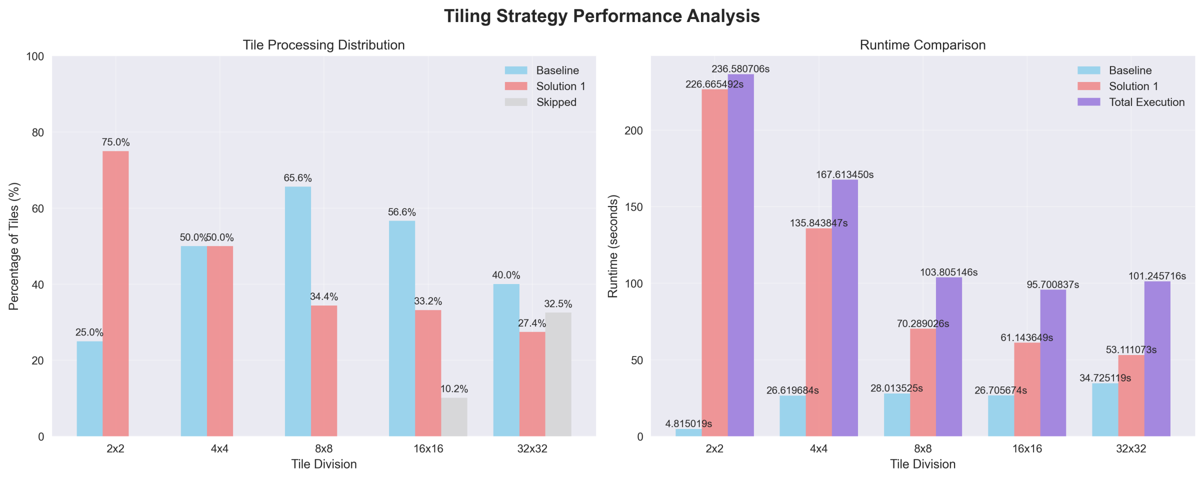


Fig. 4. Tile division analysis of tile 18_43

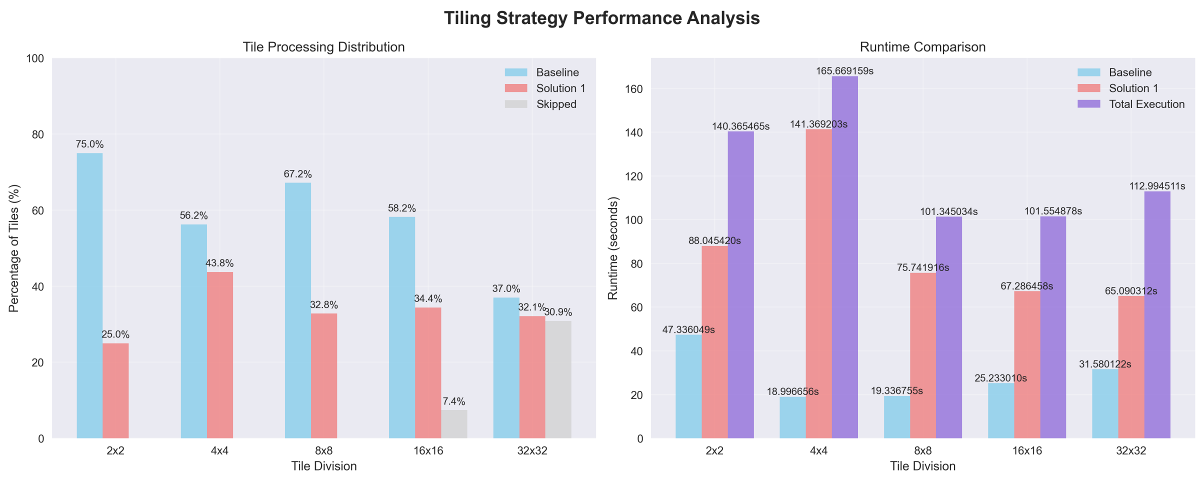


Fig. 5. Tile division analysis of tile 20_50

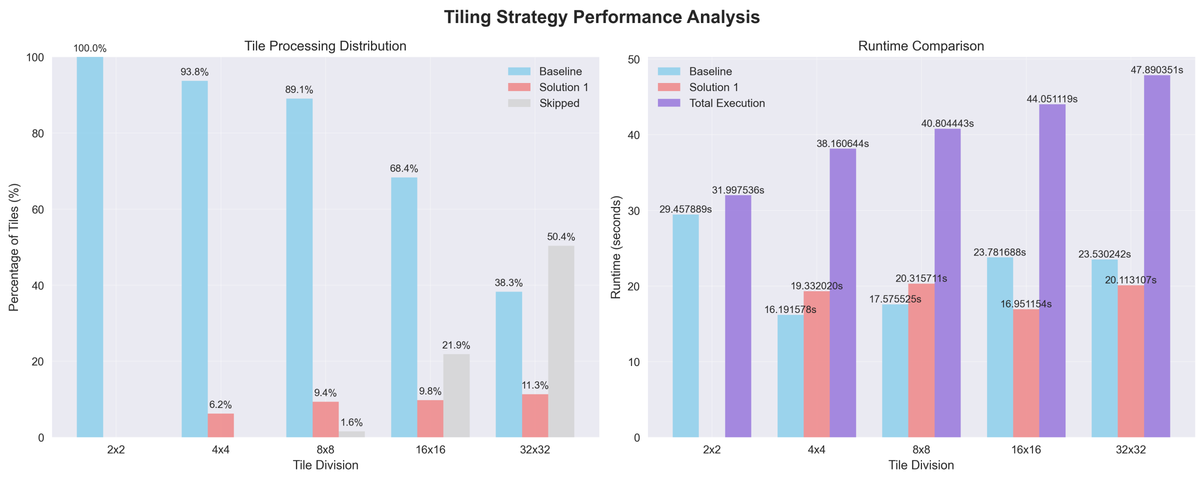


Fig. 6. Tile division analysis of tile 24_52

A.3 Evaluation Across Different Thresholds: Detailed Plots

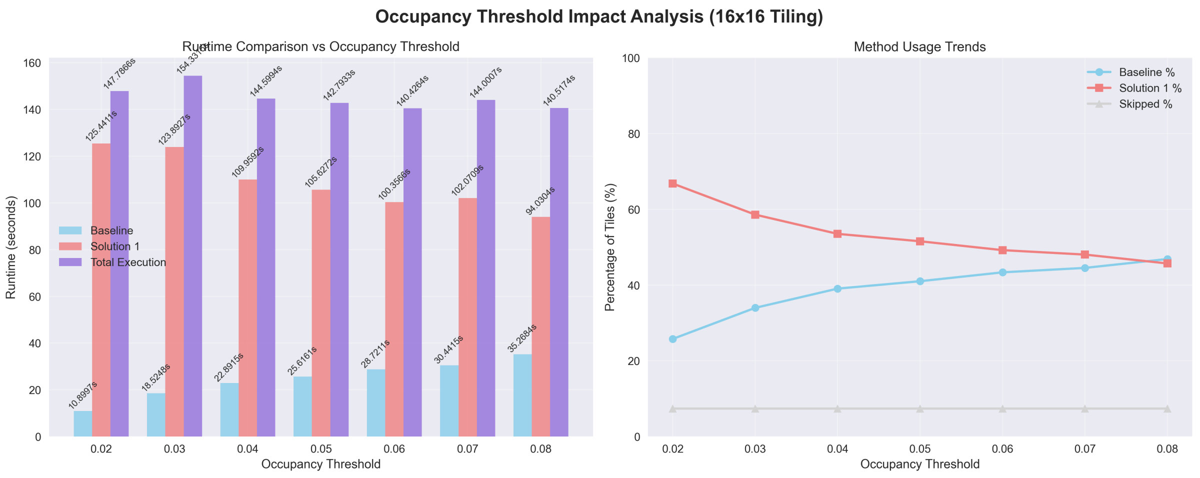


Fig. 7. *OccupancyRate* threshold analysis of tile 18_42

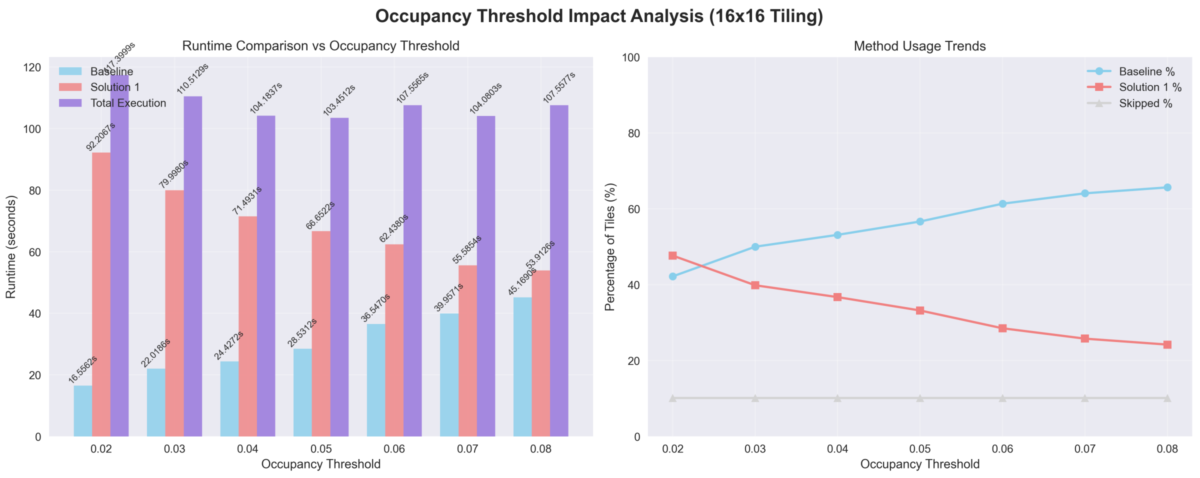


Fig. 8. *OccupancyRate* threshold analysis of tile 18_43

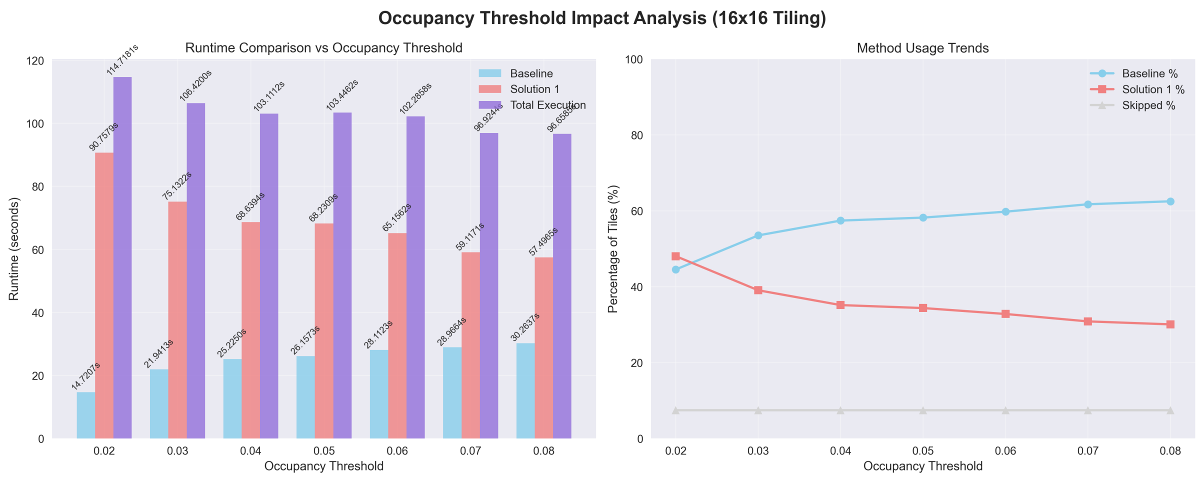


Fig. 9. *OccupancyRate* threshold analysis of tile `20_50`

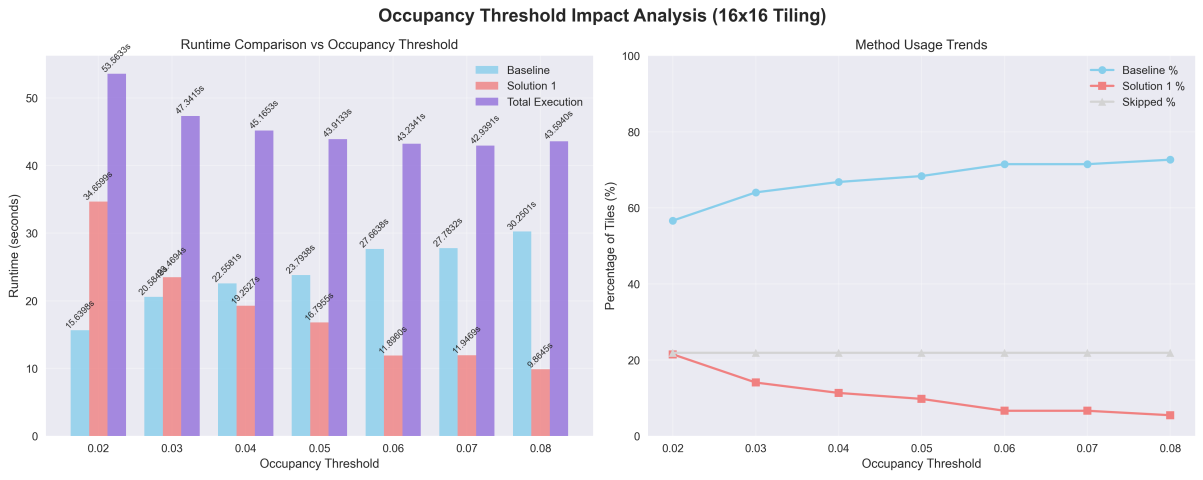


Fig. 10. *OccupancyRate* threshold analysis of tile 24_52