Performance Comparison: Fast Copy vs. Normal Copy

1. Overview

The objective is to develop a performance-critical data movement routine for a simulated execution environment that models strict instruction timing. The primary goal is to move data from a source buffer to a destination while minimizing "control checks"—the overhead created when a computer evaluates a loop condition.

2. Implementation Codes

Normal Copy (Standard Iteration)

This method uses a standard counting loop. It is simple but introduces a control check for every single element transferred.

```
def normal_copy(source, count):
    destination = [None] * count
    for i in range(count):
        destination[i] = source[i]
    return destination
```

Fast Copy (Duff's Device / Loop Unrolling)

This method reduces overhead by processing data in fixed-width groups of exactly eight elements.

```
def fast_copy(source, count):
    destination = [None] * count
    n = (count + 7) // 8
    remainder = count % 8
    source_idx = 0
    dest_idx = 0

    start_count = remainder if remainder > 0 else 8
    if start_count >= 8:
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
    if start_count >= 7:
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
    if start_count >= 6:
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
    if start_count >= 5:
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
    if start_count >= 4:
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
    if start_count >= 3:
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
    if start_count >= 2:
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
    if start_count >= 1:
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
```

```
    n -= 1

    while n > 0:
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
        destination[dest_idx] = source[source_idx]; dest_idx += 1; source_idx += 1
        n -= 1

    return destination
```

3. Comparison of Execution Logic

Feature	**Normal Copy**	**Fast Copy**
Logic Type	Compact Iteration	Loop Unrolling
Control Checks	1 check per transfer	1 check per 8 transfers
Implementation	Standard for-loop	If-sequence and while-loop
Efficiency	High overhead per element	Optimized for instruction timing

4. Execution Time Analysis

- **Simulated Environment**: In environments that model strict instruction timing, Fast Copy is significantly faster. By reducing loop condition evaluations by a factor of eight, it eliminates unnecessary branching overhead.
- **Standard Environment**: In a standard Python environment on a modern PC, the execution time for both may appear nearly identical. This is because high-level language overhead and CPU features like branch prediction neutralize the benefits of manual unrolling. However, in low-level simulated systems, the reduction in instruction count makes Fast Copy the superior choice.

Based on the research using AI tools – It Tested both programs with:

- `count = 1,000,000`
- Each function executed **5 times**
- Measured using `timeit`

Program	**Total time (5 runs)**	**Avg per run**
`normal_copy`	**0.184 s**	**~0.037 s**
`fast_copy` (Duff-style)	**0.417 s**	**~0.083 s**

Aspect	Duff's Device	Pythonic Approach
Designed for	C (low-level)	High-level
Loop overhead reduction	Effective	Minimal
Readability	Poor	Excellent
Maintainability	Hard	Easy
Performance gain	Significant in C	Negligible in Python
Recommended in Python	❌ No	✅ Yes

5. Conclusion

Normal Copy is preferred for general readability, whereas Fast Copy is a specialized optimization for low-level performance. It handles remainders via structured control flow and maximizes throughput by grouping assignments.