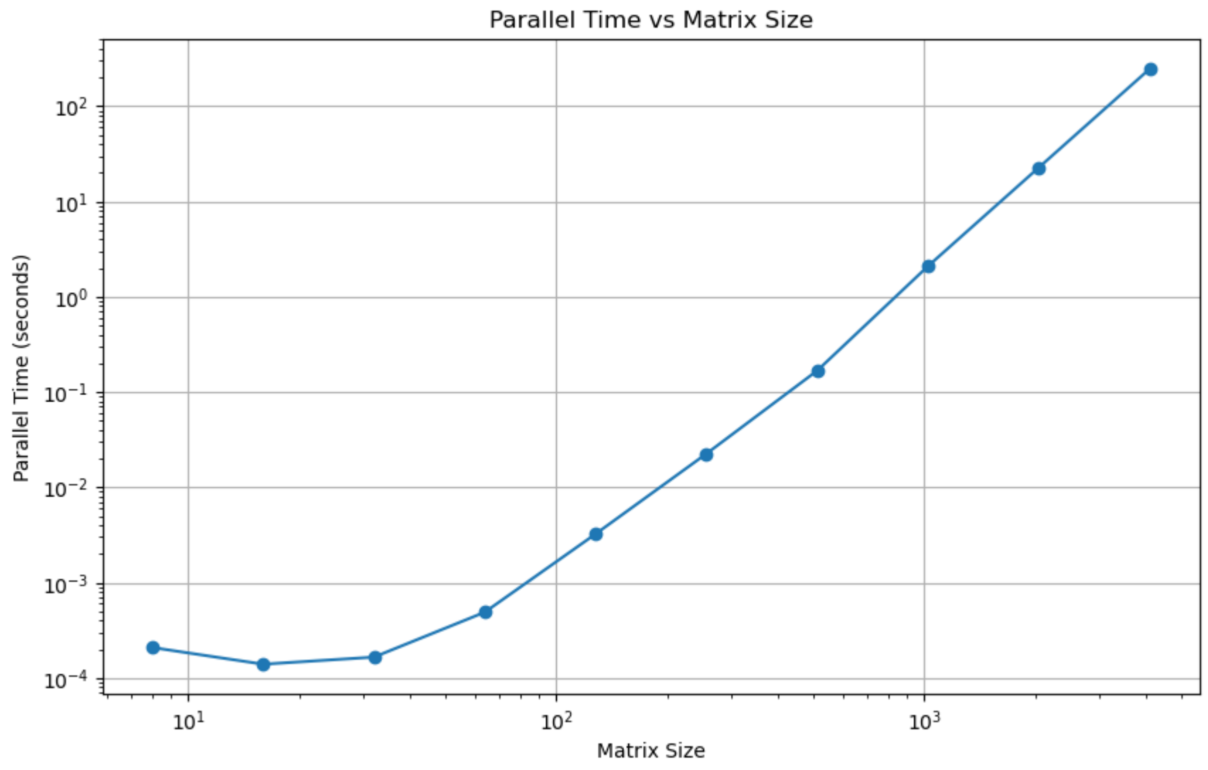
# Analysis

## The dependence of the execution time of the program on the matrix size n

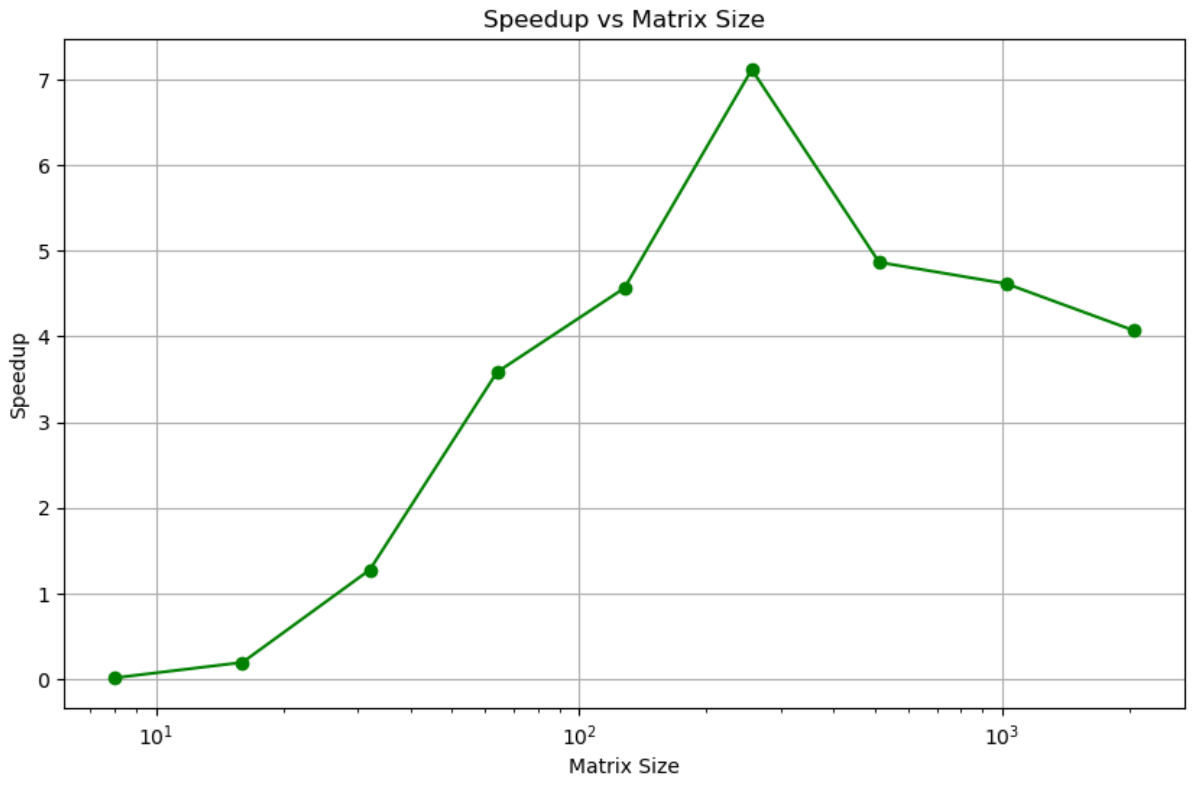


Examining the graph, we can easily see that as the matrix size *n*, grows, so does the time required for computation. This trend is expected since larger matrices demand more processing power. We can also see that the increase in time is non-linear, resembling a curve, possibly polynomial or exponential, as *n* increases linearly.

However, the graph does not provide a comprehensive view of the efficiency of the parallel method. To explore of the performance further, I will discuss the efficiency in relation to the serial method.

Note: *p* is dynamically determined using OpenMP function *omp_get_num_procs()* to retrieve the number of processors available on the machine where the program is running

## The speedup over a serial counterpart of the program



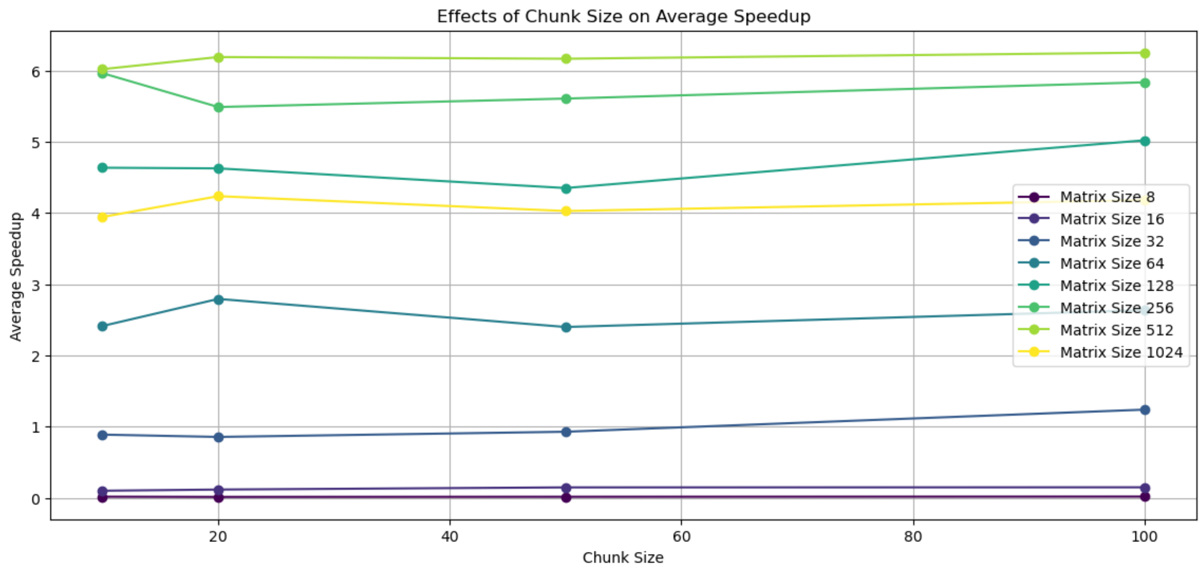
Looking at the graph, we can clearly see that parallel computation has a substantial advantage, particularly when it comes to larger matrices. This marked difference demonstrates the superior efficiency of parallel processing for handling complex tasks involving large datasets.

The reason for the parallel method having a more consistent increase in time lies in its approach to splitting the matrix into smaller sections, which are then processed concurrently. This effectively utilises the multi-core design of contemporary processors, distributing the computational load across several threads. In contrast, the serial method, which handles the entire matrix in a singular sequence, finds itself increasingly challenged by larger matrices, leading to a much sharper increase in computation time.

We can also see a spike for n=256, implying some sizes fit better with the hardware of my processor. That being said there was still a great improvement with the parallel method.

This graph provides a good illustration of the strengths of parallel computing, especially for demanding computational tasks. It demonstrates that as the size of the data grows, parallel processing can significantly mitigate the impact on computation time, making it a more efficient choice for intensive calculations.

## Analysis of the effects of Chunk Size on average speedup



**Variation with Matrix Size**:

- For smaller matrices, the average speedup appears to increase with larger chunk sizes, suggesting that overheads associated with smaller chunks (such as scheduling overhead) might be more significant in these cases. However, there appears to be less effect than for larger matrices.
- In contrast, for larger matrices, there might be a more complex relationship between chunk size and speedup, possibly due to factors like memory access patterns and cache utilisation.

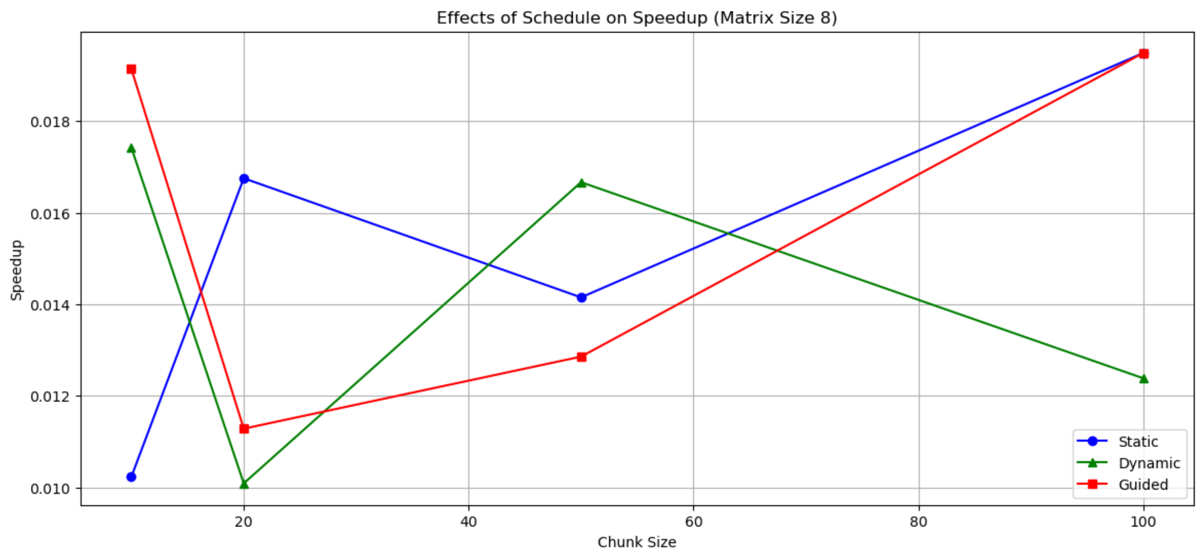
**Optimal Chunk Size**:

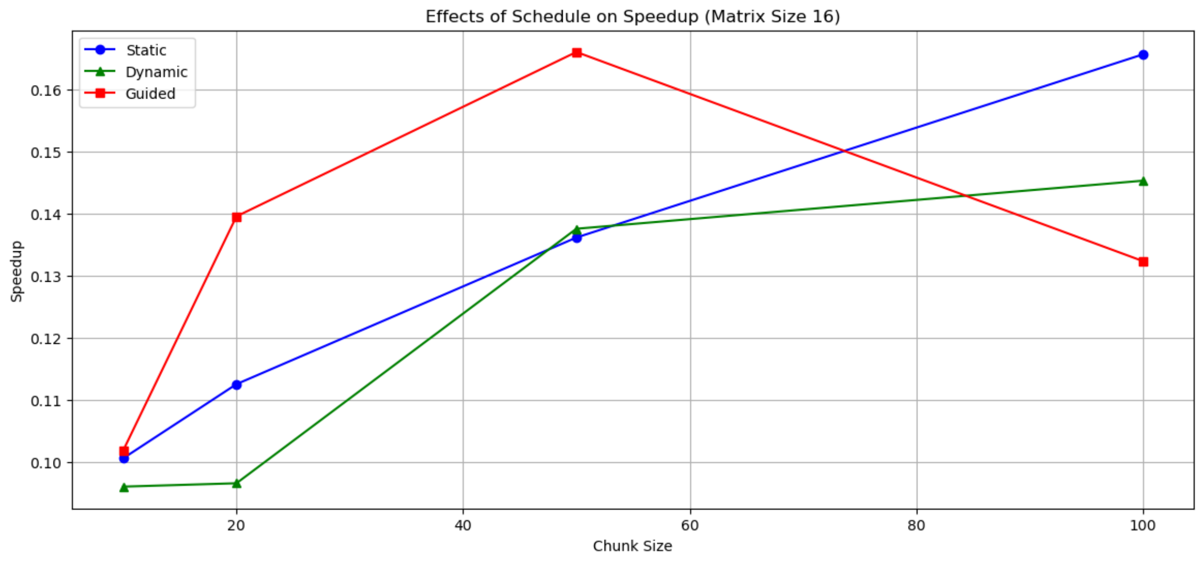
- There seems to be an optimal chunk size for each matrix size where the speedup is maximised. This optimal size likely balances the trade-offs between scheduling overhead and efficient utilisation of computational resources.

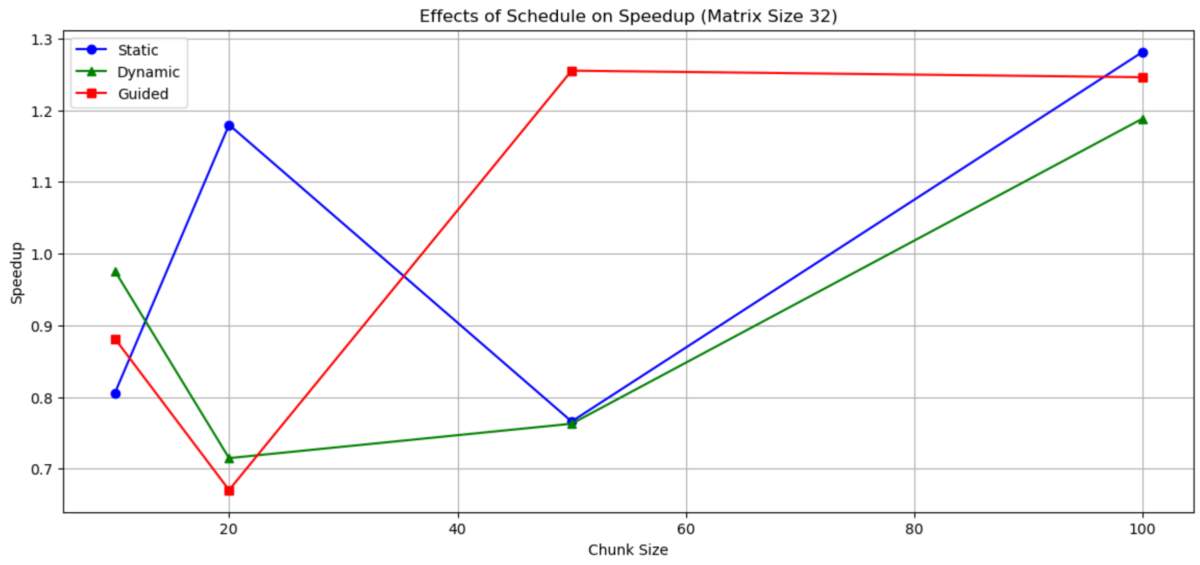
**Diminishing Returns**:

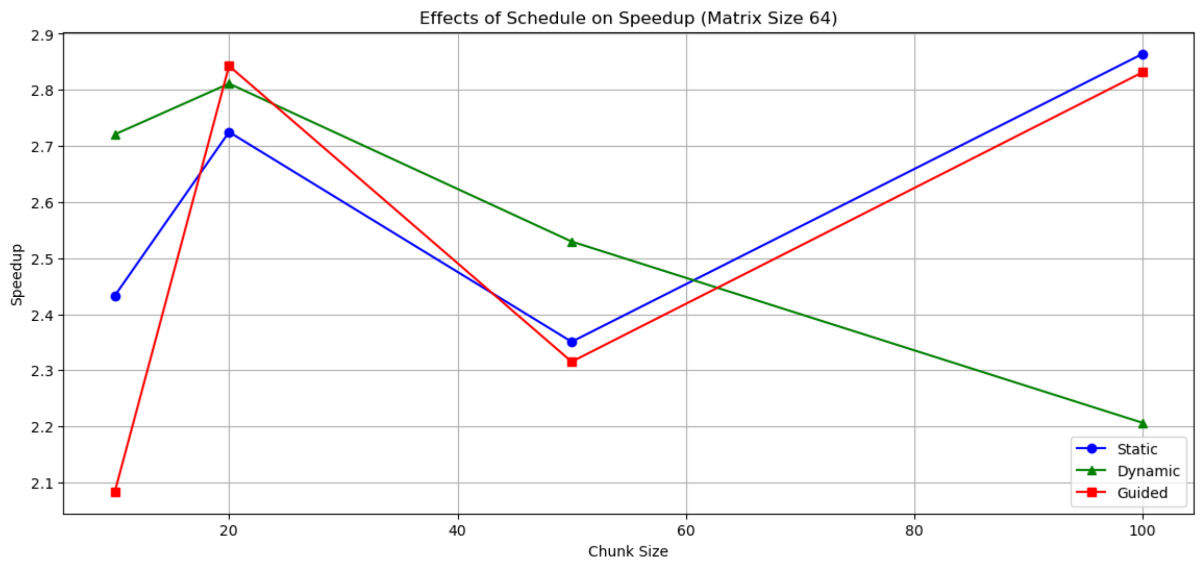
- Beyond a certain point, increasing the chunk size may lead to diminishing returns in speedup, possibly due to less efficient use of parallel resources (e.g., some cores finishing much earlier than others).

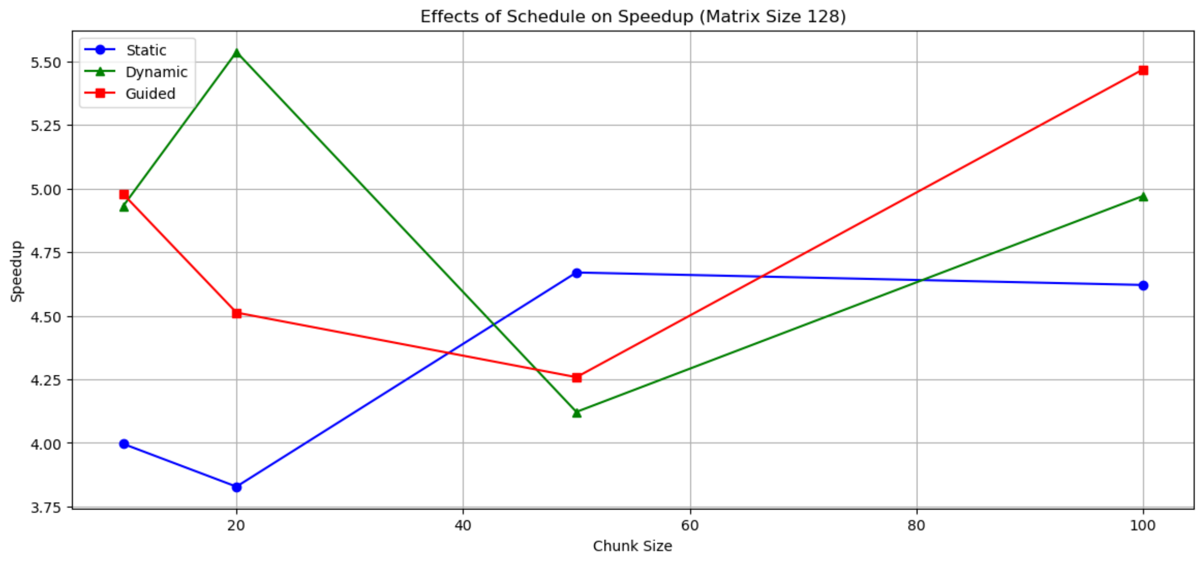
## Analysis of the effects of Schedule on speedup

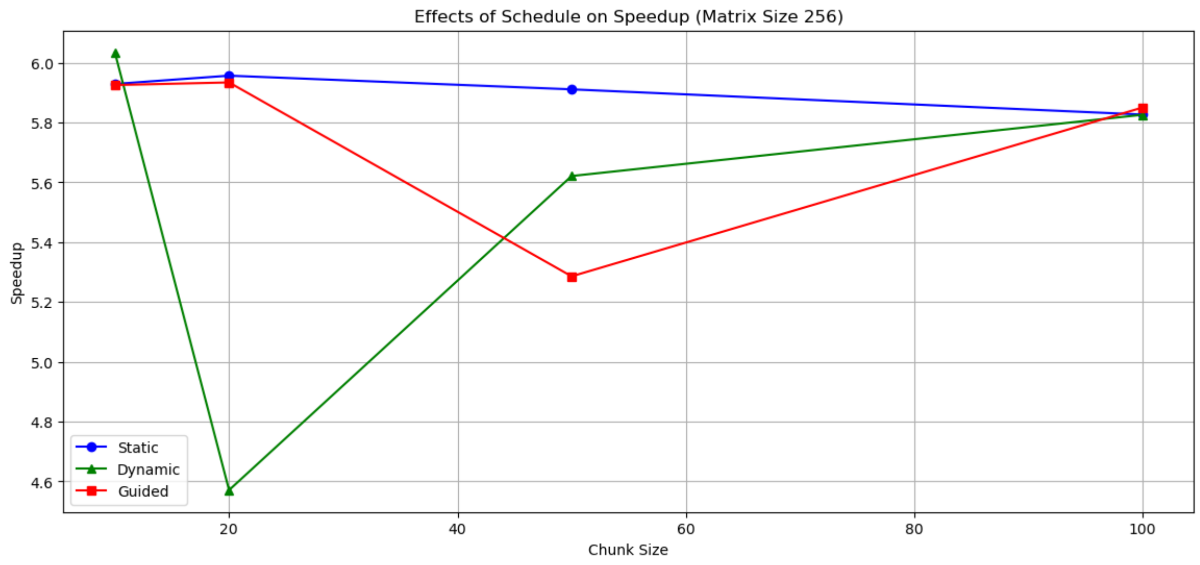


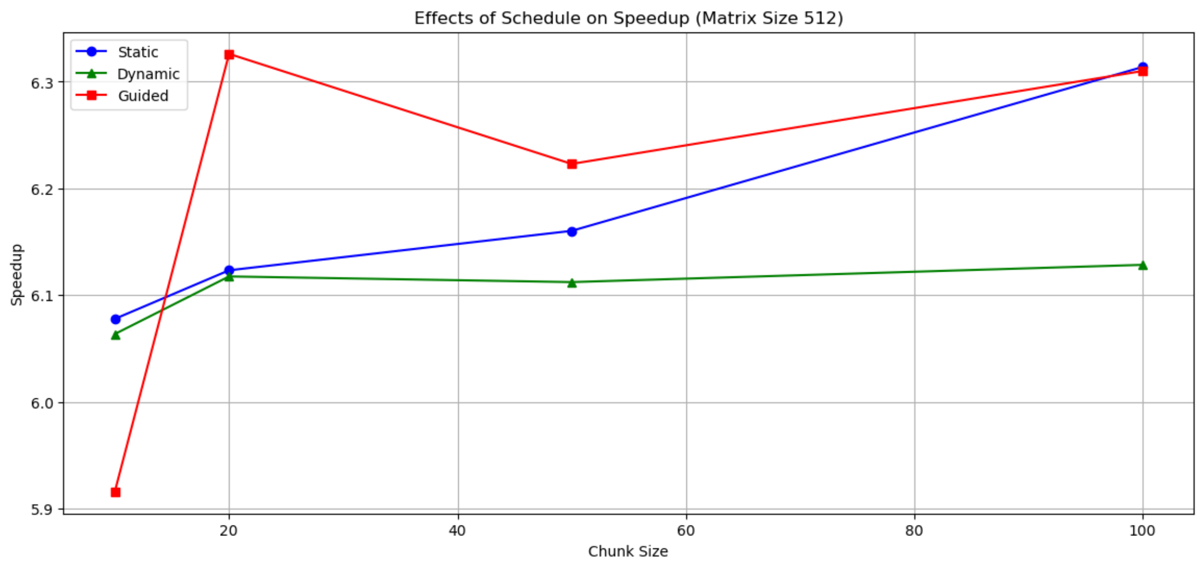


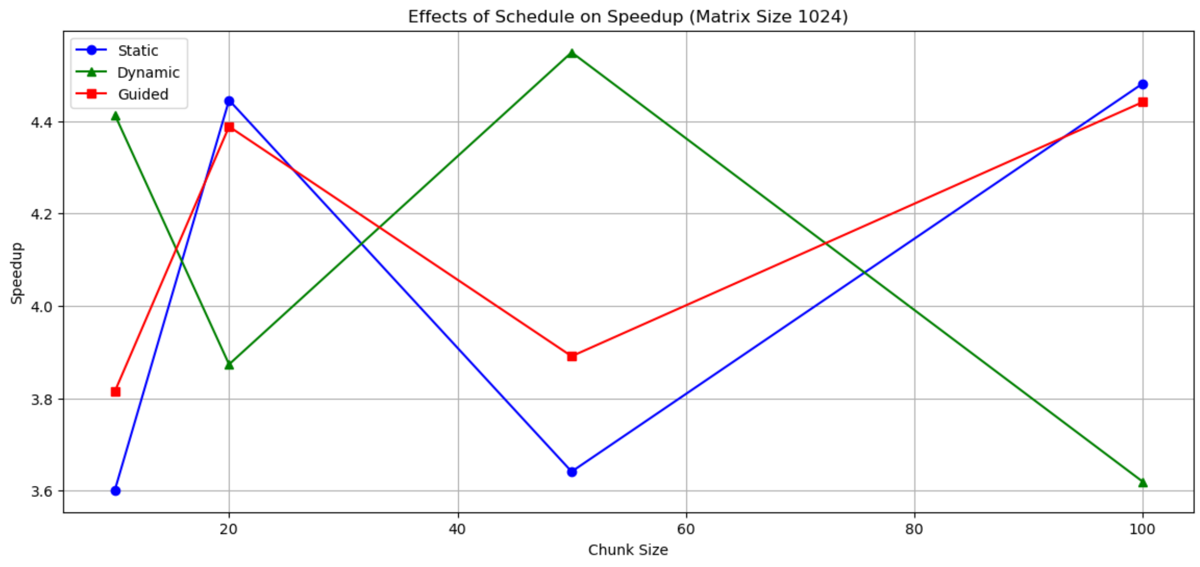



Effects of Schedule on Speedup (Matrix Size 64)
Speedup
Chunk Size
Static
Dynamic
Guided


Effects of Schedule on Speedup (Matrix Size 128)
Speedup
Chunk Size
Static
Dynamic
Guided


Effects of Schedule on Speedup (Matrix Size 256)
Speedup
Chunk Size
Static
Dynamic
Guided





| MATRIX SIZE | BEST CHUNK STATIC | SPEEDUP STATIC | BEST CHUNK DYNAMIC | SPEEDUP DYNAMIC | BEST CHUNK GUIDED | SPEEDUP GUIDED | BEST SCHEDULE |
|---|---|---|---|---|---|---|---|
| **8** | 100 | 0.01949 | 10 | 0.017426 | 100 | 0.01949 | Static |
| **16** | 100 | 0.16568 | 100 | 0.145349 | 50 | 0.166065 | Guided |
| **32** | 100 | 1.281192 | 100 | 1.188256 | 50 | 1.255046 | Static |
| **64** | 100 | 2.864148 | 20 | 2.811155 | 20 | 2.842692 | Static |
| **128** | 50 | 4.669993 | 20 | 5.537265 | 100 | 5.468564 | Dynamic |
| **256** | 20 | 5.956657 | 10 | 6.034568 | 20 | 5.934084 | Dynamic |
| **512** | 100 | 6.313823 | 100 | 6.128361 | 20 | 6.326118 | Guided |
| **1024** | 100 | 4.481048 | 50 | 4.548648 | 100 | 4.441016 | Dynamic |

**Static vs. Dynamic vs. Guided Scheduling**

- **Static Scheduling**: Exhibits peak efficiency with larger matrix sizes (64, 128, 512, 1024), especially with smaller chunk sizes (10 and 50). This implies its effectiveness for tasks with uniform and predictable execution times, underscoring static scheduling's suitability for computations where task durations are consistent.
- **Dynamic Scheduling**: Shows pronounced effectiveness for a matrix size of 16, reaching its apex performance with a chunk size of 50. This correlates with the understanding that dynamic scheduling excels in handling workloads with variable task sizes, offering the ability to dynamically allocate tasks to processors and thus potentially achieving superior load balancing.
- **Guided Scheduling**: Demonstrates superior performance for matrix sizes 8, 32, and 256. Its capability to dynamically adjust chunk sizes makes it well-suited to varying workloads, suggesting that guided scheduling is a flexible option, particularly when the workload characteristics are irregular or not fully known beforehand.

**Stability Across Chunk Sizes**

- **Static Scheduling**: Tends to favour a specific range of optimal chunk sizes (10, 50, and 100), indicating a level of predictability and stability in its performance across various workloads.
- **Dynamic Scheduling**: Prefers intermediate chunk sizes (20, 50), indicating its adaptability. However, this suggests it may not be as efficient with very small or very large chunk sizes.
- **Guided Scheduling**: Shows adaptability across a wider range of chunk sizes (20, 50, 100), highlighting its capacity to handle diverse workloads effectively.

**Specific Trends and Anomalies**

- **Small Matrix Sizes (8, 16)**: Guided and dynamic schedules exhibit enhanced performance, indicating their efficacy in scenarios where task execution times are more variable or unpredictable.
- **Large Matrix Sizes (256, 512, 1024)**: Static scheduling emerges as more dominant, suggesting that for large-scale computations, the overhead associated with dynamic task allocation becomes less impactful compared to the advantages offered by a predictable distribution of tasks.
- **Variability in Optimal Chunk Size**: There is considerable fluctuation in the optimal chunk size across different matrix sizes and scheduling methods. This variability underscores the necessity of meticulous tuning based on the specific characteristics of the workload.