> *Loop optimization is a set of techniques used to improve the efficiency and performance of loops in computer programs.*

Since loops often account for a significant portion of execution time, optimizing them can have a significant impact on overall program speed.

Some common loop optimization techniques:

1. Loop Unrolling: Loop unrolling is a technique that reduces loop overhead by executing multiple iterations of the loop body in a single iteration. By reducing the number of loop iterations and reducing branching overhead, loop unrolling can improve performance, especially when combined with other optimizations like instruction pipelining and parallel execution.

2. Loop Fusion: Loop fusion involves combining multiple adjacent loops into a single loop. By eliminating redundant iterations and reducing memory accesses, loop fusion can improve cache utilization and reduce loop overhead.

3. Loop Blocking/Loop Tiling: Loop blocking, also known as loop tiling, divides a loop into smaller blocks or tiles. By operating on smaller data subsets, loop blocking improves cache utilization, reduces memory access latency, and enables better data locality.

4. Loop-Invariant Code Motion (LICM): LICM involves moving loop-invariant computations outside the loop. By eliminating redundant computations that produce the same result in each iteration, LICM reduces the computational overhead of the loop.

5. Loop Strength Reduction: Loop strength reduction replaces expensive operations within a loop with equivalent but less costly operations. For example, replacing a multiplication operation with a shift operation or replacing a division with a multiplication by an inverse.

6. Loop Parallelization: Loop parallelization techniques, such as loop-level parallelism or auto-vectorization, enable executing loop iterations concurrently on multiple processing units or SIMD (Single Instruction, Multiple Data) units. This technique leverages the parallel execution capabilities of modern processors to speed up loop execution.

7. Loop Interchange: Loop interchange swaps the order of nested loops to improve data locality and cache utilization. By accessing memory in a more contiguous manner, loop interchange can reduce cache misses and improve performance.

8. Loop Vectorization: Loop vectorization converts scalar operations within a loop into SIMD vector operations. It enables performing multiple computations simultaneously using vector registers, resulting in improved performance on architectures with SIMD capabilities.

9. Loop-Invariant Code Elimination (LICE): LICE identifies and eliminates computations that produce the same result in each loop iteration. By moving such computations outside the loop, LICE reduces redundant computations and improves performance.

10. Loop Pipelining: Loop pipelining breaks down loop iterations into smaller stages that can be executed concurrently. It enables overlapping the execution of multiple iterations, reducing the overall loop execution time.

Unoptimized code

```
repeat
S1;
i := j;
s2
until p;
```

Optimized code

```
i := j;
repeat
Sl;
s2
until p;
```