For loop line 27

```
for(int i = 0; i < SIZE; ++i) {
    printf("%d ", a[i]);
    printf("%d ", b[i]);
}
```

From the output we can read that the compiler first starts by analyzing zhe loop nest structure to understand the loop hierarchy. Next the compiler checks the loop bounds and the way the loop goes through the indexes. In === get_loop_niters === the number of total iteration of the loop will get calculated. After it did those steps it will look into the code in the loop. The conclusion of this is then that this loop cannot be vectorized because the printf "clobbers" the memory. This means that the printf function modifies the memory in some way, that is to complex for the compiler to analyze. Therefore it is not able to vecotize this for loop. (It tried with the modes V16QI, V8QI, V4QI)

For loop line 22

```
for(int i = 1; i < SIZE-1; ++i) {
        a[i] = a[i%argc];
}
```

The analysis of the loop at line 22 shows that the loop failed to be vectorized due to a bad data dependence between the array accesses. Argc is not known at the start of the program. It would be required to match the size of a register in regatd to the datatype of a to be vectorizeable.

For loop line 10

```
for(int i = 0; i < SIZE; ++i) {
        a[i] = argc;
}
```

Here we have additional analysis steps.

=== vect_analyze_data_refs === means that the compiler nalyzed the data references in the loop to understand the memory access.

=== vect_analyze_scalar_cycles === This analyze is to understand the depentency in the loop to identify loop carried dependencies.

=== vect_determine_precisions === Here the compiler tries to determin the required precision of the numbers.

=== vect_pattern_recog === Here pattern recognition is used to identify simple use patterns.

=== vect_analyze_data_ref_accesses === The compiler analyzes the data reference accesses to identify potential conflicts or dependencies.

=== vect_mark_stmts_to_be_vectorized === This step marks the relevant statements to be vectorized based on their dependencies and characteristics.

In this section we can aly read that we got an vectorizazion factor of 4. This would mean we can execute the loop 4 times in parallel. The compiler will still continue to analyze in case there is a better solution. Due to this there will be a few new analyzation steps.

=== vect_compute_single_scalar_iteration_cost === The cost of a single scalar iteration is computed to estimate the performance impact of vectorization.

=== vect_analyze_slp === The compiler performs the Straight-Line-Pattern (SLP) analysis to identify opportunities for vectorization.

=== vect_make_slp_decision === A decision is made whether to apply the SLP transformation based on the analysis results.

=== vect_analyze_data_refs_alignment === The compiler analyzes the alignment of data references to ensure efficient vectorization.

=== vect_prune_runtime_alias_test_list === The compiler prunes the runtime alias test list to optimize the vectorization process.

=== vect_enhance_data_refs_alignment === The alignment of data references is enhanced to enable better vectorization opportunities.

=== vect_dissolve_slp_only_groups === The compiler dissolves the SLP-only groups to facilitate further vectorization.

=== vect_analyze_loop_operations === The compiler analyses the loop operations and their dependencies to determine vectorization potential.

After all those steps we get a cost model anaysis with the following listing.: Vector inside of loop cost: This refers to the cost of executing vectorized instructions inside the loop body.

Vector prologue cost: The prologue refers to the code executed before the vectorized loop begins.

Vector epilogue cost: The epilogue refers to the code executed after the vectorized loop finishes.

Scalar iteration cost: This refers to the cost of executing scalar instructions within each iteration of the loop.

Scalar outside cost: The scalar outside cost represents the cost of executing scalar instructions outside the loop.

Vector outside cost: The vector outside cost represents the cost of executing vectorized instructions outside the loop.

Prologue iterations: This line indicates the number of iterations executed in the prologue of the loop.

Epilogue iterations: This line indicates the number of iterations executed in the epilogue of the loop.

Calculated minimum iters for profitability: This line represents the minimum number of loop iterations required for the loop vectorization to be considered profitable.

At the end of this block we see a "LOOP VECOTIRIZED" to indicate that a valid vectoriation has been found. We now also know we only need 1 iteration to profit from the optimisation.

Function main line 5

At the end we get a small summary about the vectorisation process. The first line indicates, that 1 loop in this function got vectorized. Line 15 could not get vectorized because it is a statement that clobbers memory. The same gose for the printf statements. The remaining loops don't have any data in the output about theire behavioure.