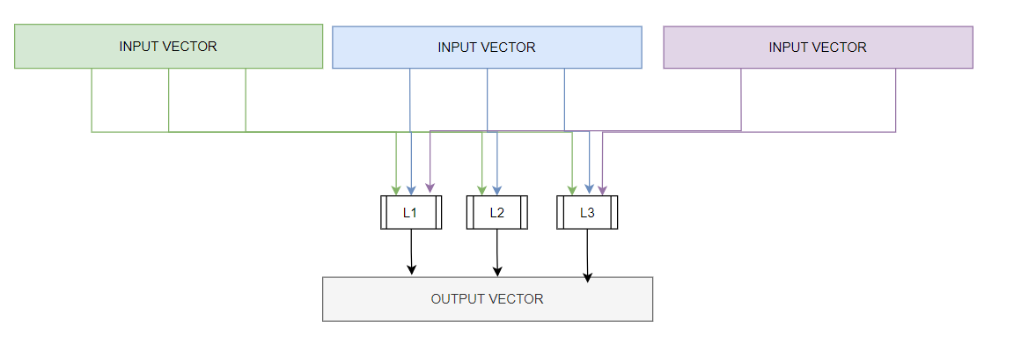
This paper discusses techniques to vectorize performance incentive portions of large-scale Java server applications using Java VectorAPIs [1]. Using these we were able to attain significant speedup in two key algorithms [2][3] used in Apache parquet-mr columnar storage library and spark analytic engine.

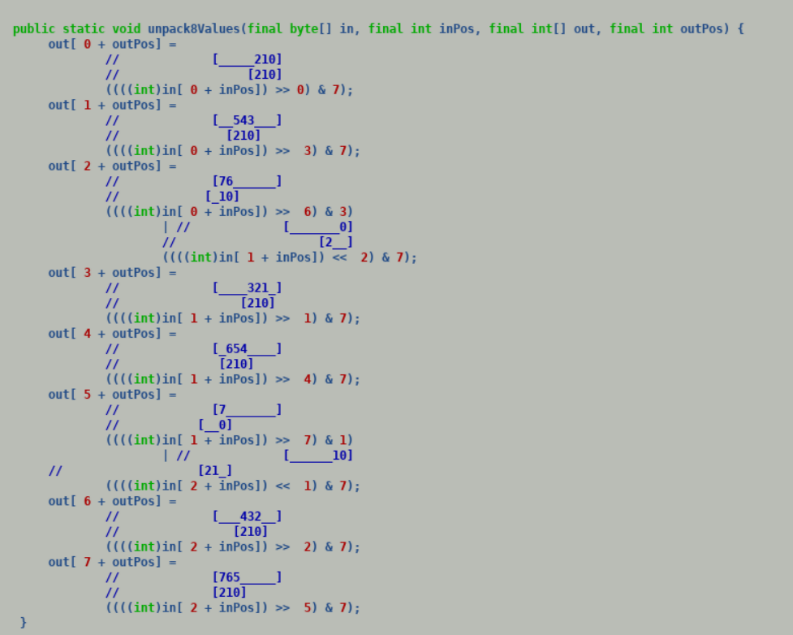
**A) Bit Unpacking algorithm heavily used during decoding stage [Fang kindly correct if needed]**

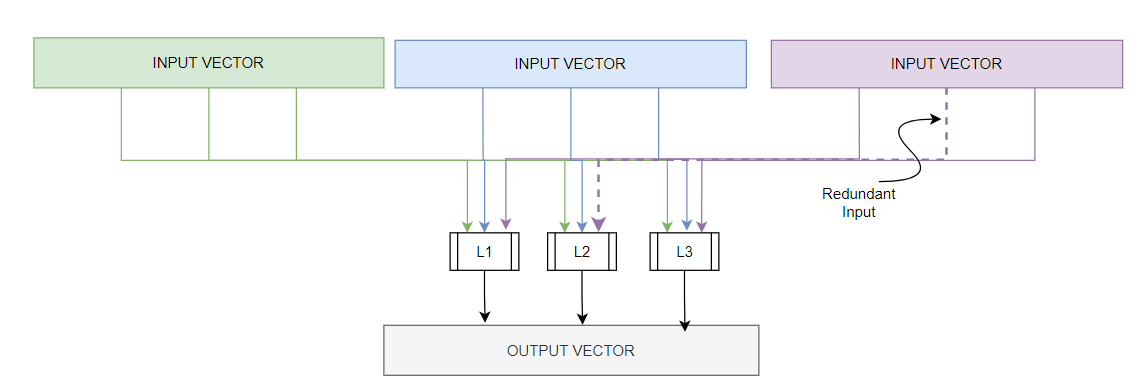
Step 1: Identify code blocks which read from or write to contiguous memory locations (non-contiguous accesses needs special vector gather / scatter operations)

Step 2: Analyze the expression trees feeding into these memory locations, bottom-up traversal of these expression trees determines the sequence in which vector operations needs to be performed. A lane wise vector operations reads the contents of source vector lanes perform designated operations over it and write the contents into corresponding lanes of destination vector. Thus, source and destination must have same lane count.

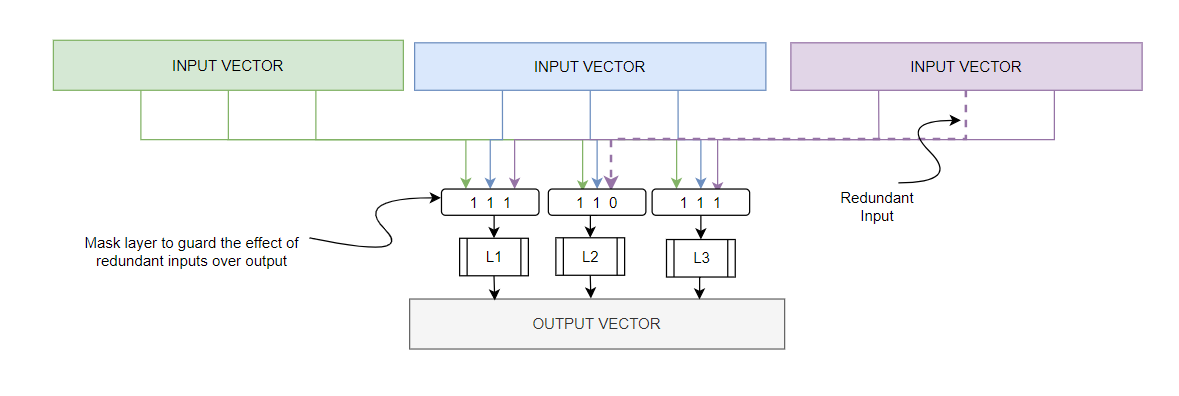


Step 3: If needed introduce redundancies to ensure shape conformance across participating scalar expressions such that they can be packed into vector operations. Following Bit unpacking algorithm comprises of multiple scalar assignments and LHS expressions of these assignments are not isomorphic, this makes it difficult to pack the corresponding elements of each expression into a vector. By introducing redundancies, we can enable structure isomorphism across all the LHS expressions.

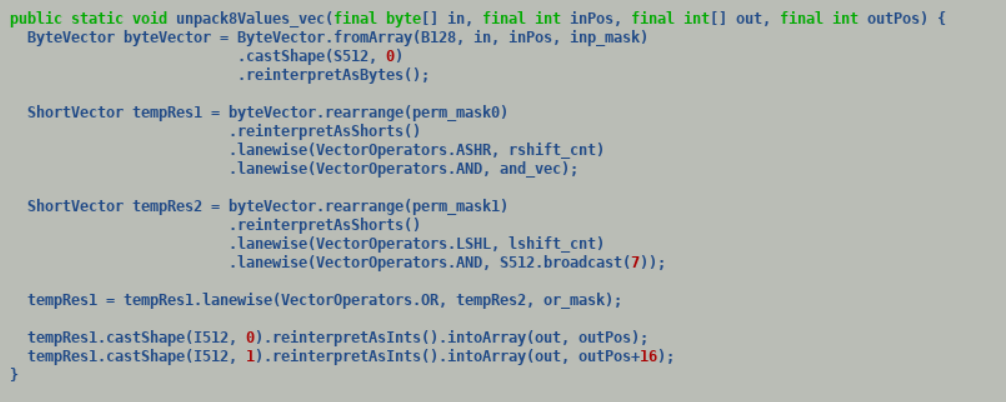




Step 4: Use predicated vector operations to guard the effect of these redundances over resultant vectors to ensure correct semantics.



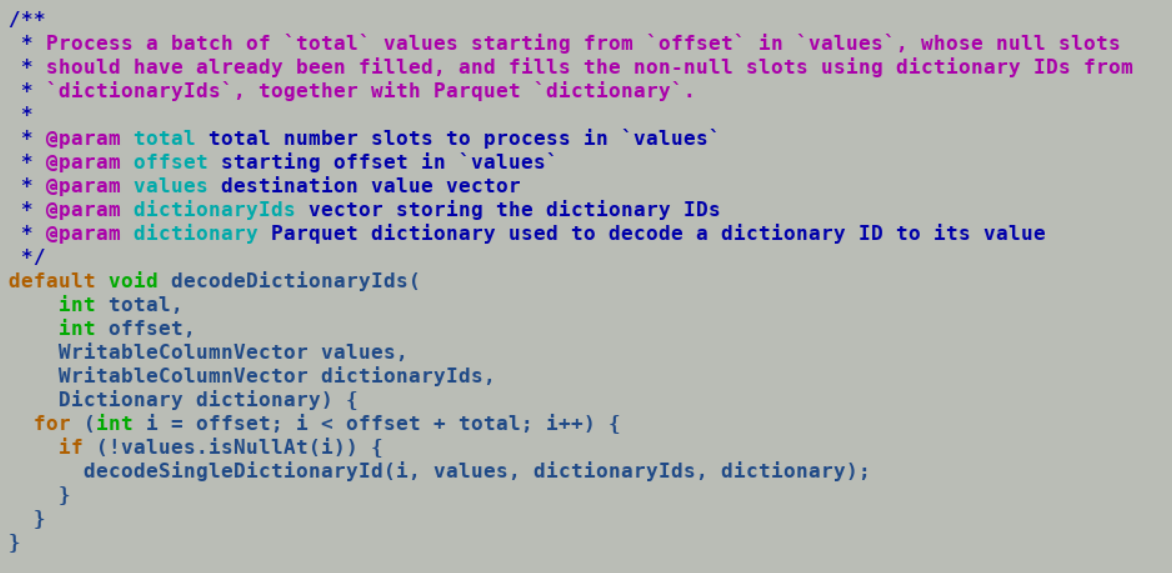
Step 5: Scalar up the vector algorithm by choosing the maximal possible vector SPECIES supported by the target. Following code shows the final version of vectorized bit packing algorithm.



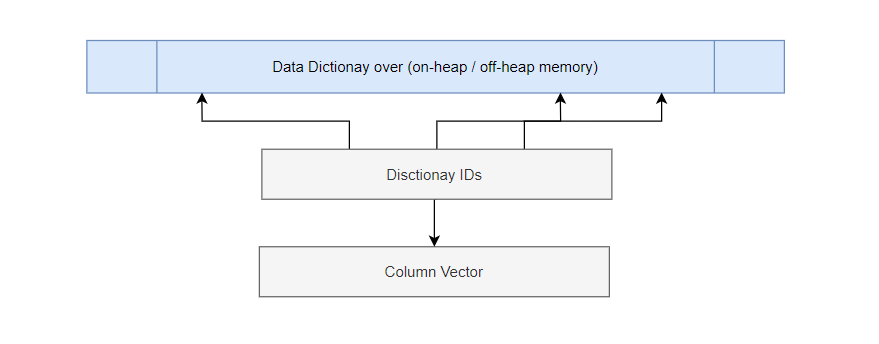
Using above technique, we were able to showcase 5-10x speedup over existing bit backing algorithm through micro-benchmarks. We are in process of enhancing parquet-mr code generators to emit vectorized bit-packers to accelerate decoding process.

B) Dictionary Decoding:

Dictionary decoding algorithm is used during batch read operation to populate column vectors. It reads the contents of data dictionary from locations specified by dictionary ids and stores the contents into a column vector. Since dictionary ids can correspond to non-contiguous locations hence this operation directly mimics a gather operation.



Parquet column vectors are either backed by off heap or on heap memory. Off heap memory is out of purview of JVM and is not influenced by GCs, existing parquet APIs reads and write one primitive value (byte/char/int/long) at a time during batch read operation, this can be made performant by storing the underlined off-heap memory over MemorySegments[4] which allows memory access at granularity of a vector size using VectorAPI memory segment access APIs planned to be shipped with upcoming incubator 4 release[5]. On heap memory is backed by primitive arrays is part of JVM managed memory. VectorAPI gather [6] operation directly facilitate loading the contents of non-contiguous memory into vector.



[ Fang please provide performance stats of micro benchmarks]

References:

[1]<https://docs.oracle.com/en/java/javase/18/docs/api/jdk.incubator.vector/jdk/incubator/vector/package-summary.html>

[2] Bit Unpacking generator: <https://github.com/apache/parquet-mr/blob/master/parquet-generator/src/main/java/org/apache/parquet/encoding/bitpacking/IntBasedBitPackingGenerator.java#L129>

[3]Dictionary Decoding: <https://github.com/apache/spark/blob/master/sql/core/src/main/java/org/apache/spark/sql/execution/datasources/parquet/ParquetVectorUpdater.java#L67>

[4] Foreign Memory Segments: <https://docs.oracle.com/en/java/javase/18/docs/api/jdk.incubator.foreign/jdk/incubator/foreign/MemorySegment.html>

[5] VectorAPIs for MemorySegment access: <https://bugs.openjdk.java.net/browse/JDK-8280173>

[6] Gather API:<https://docs.oracle.com/en/java/javase/18/docs/api/jdk.incubator.vector/jdk/incubator/vector/IntVector.html#fromArray(jdk.incubator.vector.VectorSpecies,int%5B%5D,int,int%5B%5D,int,jdk.incubator.vector.VectorMask)>