You can find the part that we are mainly expected to optimize from line 171 of `mttkrp.c`:

```
sptStartTimer(timer);
for(sptNnzIndex x=0; x<nnz; ++x) {
    mode_i = mode_ind[x];
    tmp_i_1 = times_inds_1[x];
    tmp_i_2 = times_inds_2[x];
    entry = vals[x];

    for(sptIndex r=0; r<R; ++r) {
        mvals[mode_i * stride + r] += entry * times_mat_1->values[tmp_i_1 * stride + r] *
times_mat_2->values[tmp_i_2 * stride + r];
    }
}
sptStopTimer(timer);
```

Here `nnz` is the number of non-zero elements of the given tensor i.e. `nell-2.tns`. In the outer loop, we iterate every non-zero element of the given tensor and do some related calculation. `(mode_i, tmp_i_1, tmp_i_2)` is this element's position in the tensor. If the size of the tensor is $n \times m \times p$, then we have $1 \leq \text{mode\_i} \leq n, 1 \leq \text{tmp\_i\_1} \leq m, 1 \leq \text{tmp\_i\_2} \leq p$. For `nell-2.tns`, n, m, p equal to 12092, 9184 and 28818 respectively. At the same time, `entry` is the value of this element which is non-zero.

For convenience, below we use i, j, k to refer to `(mode_i, tmp_i_1, tmp_i_2)` respectively. By the way, `sptNnzIndex` and `sptIndex` are all scalar types like `int`, so we do not need to pay much attention to it.

In the inner loop, `mvals`, `times_mat_1->values`, and `times_mat_2->values` are all matrixes explicitly stored in an 1-dimension array, so their types are all `sptValue*` where `sptValue` is actually `float` whose size if 4 bytes under default configuration. Their store pattern is row-first. The row number of `mvals`, `times_mat_1->values` and `times_mat_2->values` are $\max\{n, m, p\}$, m, and p respectively, whereas their column number are all equal to R. R can be adjusted by command line argument `-r` of `mttkrp` program(it is ceiled to a multiple of 8), and it is 16 by default. `stride` is equal to R, so it is also 16 by default. When it comes to initial values of these matrixes, `mvals` is the output therefore it is all zero be default, but other two matrixes are initialized with random values.

We use M^i to represent row i of matrix M. Then, what we do in the inner loop can be concluded to be $\text{mvals}^i := \text{mvals}^i + \text{entry} \times \text{mat}_1^j \cdot \text{mat}_2^k$. Here $\cdot$ is a kind of vector multiplication. For example, the result of $[1, 2, 3]^T \cdot [1, 2, 3]^T$ equal to $[1 \times 1, 2 \times 2, 3 \times 3]^T$ that is $[1, 4, 9]^T$.

Overall, the whole thing what we should do is that for all non-zero elements of the given tensor whose position is (i, j, k) and value is `entry`, do $\text{mvals}^i := \text{mvals}^i + \text{entry} \times \text{mat}_1^j \cdot \text{mat}_2^k$.

What should be mentioned that above analysis are base on `mode=0`. When `mode` varies we only need to swap n, m, p and (i, j, k), basically we are still doing the same thing.