```c
multiReturn matrixMultiOMP(matrix matrixA, matrix matrixB, matrix matrixC){
   printf("\nStart parallel multiplication with openMP v1 ... \n");

   double time1=omp_get_wtime();
   #pragma omp parallel //use omp parallelisation
   {

     #pragma omp for private(col,add) // col and add are privat every thread get its own var.
         (row=0;row<matrixA.rows;row++){
            (col=0;col<matrixB.cols;col++){
           int tempSum=0;
              (add=0;add<matrixA.cols;add++){
              tempSum += matrixA.matrix[row][add] * matrixB.matrix[add][col];
          }
          matrixC.matrix[row][col] = tempSum;
        }
      }
   }
   double time2=omp_get_wtime();
   printf("finished\n\n");

   multiReturn mr = {"OMP",time2-time1}; // datatype for return

         mr;
}


// second version for omp multiplication not used in programm is a little slower

multiReturn matrixMultiOMP2(matrix matrixA, matrix matrixB, matrix matrixC){
   printf("\nStart parallel multiplication with openMP v2 ... \n");

   double time1=omp_get_wtime();

         (row=0;row<matrixA.rows;row++){
         #pragma omp parallel
         {
         #pragma omp for private(add)
               (col=0;col<matrixB.cols;col++){
             int tempSum=0;
                (add=0;add<matrixA.cols;add++){
                tempSum += matrixA.matrix[row][add] * matrixB.matrix[add][col];
             }
             matrixC.matrix[row][col] = tempSum;
          }
        }
   }
   double time2=omp_get_wtime();
   printf("finished\n\n");

   multiReturn mr = {"OMP2",time2-time1};

         mr;
}

// third version for omp multiplication not used in programm is a much more slower  then v1
// also much more slower then sequential mulitplication

multiReturn matrixMultiOMP3(matrix matrixA, matrix matrixB, matrix matrixC){
   printf("\nStart parallel multiplication with openMP v3 ... \n");

   double time1=omp_get_wtime();

         (row=0;row<matrixA.rows;row++){
               (col=0;col<matrixB.cols;col++){
             int tempSum=0;
             #pragma omp parallel for reduction(+:tempSum) // tempsum = critical area
                (add=0;add<matrixA.cols;add++){
                tempSum += matrixA.matrix[row][add] * matrixB.matrix[add][col];
             }
             matrixC.matrix[row][col] = tempSum;
```

```c
        }
    }
    double time2=omp_get_wtime();
    printf("finished\n\n");

    multiReturn mr = {"OMP3",time2-time1};

            mr;
}
```