```c
#include <errno.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MIN(a, b) ((a) < (b) ? (a) : (b))
#define CELL(mat, row, col) ((mat)->data[(row) * (mat)->n + (col)])
#define CSV_DELIM ","

typedef struct matrix_struct {
    size_t m;
    size_t n;
    float data[];
} matrix_t;

typedef int matmul_t(matrix_t *a, matrix_t *b, matrix_t **out_result, size_t b_size);

matrix_t *make_matrix(size_t m, size_t n) {
    matrix_t *result = calloc(sizeof(*result) + m * n * sizeof(*result->data), 1);
    if (!result) {
        return NULL;
    }
    result->m = m;
    result->n = n;
    return result;
}

int read_matrix(const char *path, matrix_t **out_result) {
    FILE *file = NULL;
    matrix_t *result = NULL;
    int ret = -EINVAL;

    file = fopen(path, "r");
    if (!file) {
        goto cleanup;
    }

    size_t m;
    size_t n;
    if (fscanf(file, " %zu x %zu ", &m, &n) != 2) {
        goto cleanup;
    }

    result = make_matrix(m, n);
    if (!result) {
        goto cleanup;
    }

    for (size_t i = 0; i < m; i++) {
        if (fscanf(file, " %f ", &CELL(result, i, 0)) != 1) {
            goto cleanup;
        }
        for (size_t j = 1; j < n; j++) {
            if (fscanf(file, " " CSV_DELIM "%f ", &CELL(result, i, j)) != 1) {
                goto cleanup;
            }
        }
    }

    *out_result = result;
    ret = 0;

cleanup:
    if (file) {
        fclose(file);
    }
    if (ret) {
        free(result);
    }
    return ret;
}

void write_matrix(matrix_t *matrix, FILE *file) {
    fprintf(file, "%zux%zu\n", matrix->m, matrix->n);
    for (size_t i = 0; i < matrix->m; i++) {
        fprintf(file, "%f", CELL(matrix, i, 0));
        for (size_t j = 1; j < matrix->n; j++) {
            fprintf(file, ",%f", CELL(matrix, i, j));
        }
        fprintf(file, "\n");
    }
}

#define DEFINE_PLAIN_MATMUL(lp0, lp1, lp2)                                              \
    int plain_matmul_##lp0##lp1##lp2(matrix_t *a, matrix_t *b, matrix_t **out_result) { \
        if (a->n != b->m) {                                                             \
            return -EINVAL;                                                             \
        }                                                                               \
                                                                                        \
        size_t m = a->m;                                                                \
        size_t n = b->n;                                                                \
        size_t k = a->n;                                                                \
                                                                                        \
        matrix_t *result = make_matrix(m, n);                                           \
        if (!result) {                                                                  \
            return -ENOMEM;                                                             \
        }                                                                               \
                                                                                        \
        result->m = m;                                                                  \
        result->n = n;                                                                  \
                                                                                        \
        size_t i_end = m;                                                               \
        size_t j_end = n;                                                               \
        size_t p_end = k;                                                               \
                                                                                        \
        clock_t measurement_begin = clock();                                            \
                                                                                        \
        for (size_t lp0 = 0; lp0 < lp0##_end; lp0++) {                                  \
            for (size_t lp1 = 0; lp1 < lp1##_end; lp1++) {                              \
                for (size_t lp2 = 0; lp2 < lp2##_end; lp2++) {                          \
                    CELL(result, i, j) += CELL(a, i, p) * CELL(b, p, j);                \
                }                                                                       \
            }                                                                           \
        }                                                                               \
                                                                                        \
        printf("%ld\n", clock() - measurement_begin);                                   \
                                                                                        \
        *out_result = result;                                                           \
        return 0;                                                                       \
    }

#define DEFINE_BLOCK_MATMUL(lp0, lp1, lp2)                                                             \
    int block_matmul_##lp0##lp1##lp2(matrix_t *a, matrix_t *b, matrix_t **out_result, size_t b_size) { \
        if (a->n != b->m) {                                                                            \
            return -EINVAL;                                                                            \
        }                                                                                              \
                                                                                                       \
        size_t m = a->m;                                                                               \
        size_t n = b->n;                                                                               \
        size_t k = a->n;                                                                               \
                                                                                                       \
        matrix_t *result = make_matrix(m, n);                                                          \
        if (!result) {                                                                                 \
            return -ENOMEM;                                                                            \
        }                                                                                              \
                                                                                                       \
        result->m = m;                                                                                  \
        result->n = n;                                                                                  \
                                                                                                        \
        clock_t measurement_begin = clock();                                                            \
                                                                                                        \
        for (size_t ii = 0; ii < m; ii += b_size) {                                                     \
            size_t ib_size = MIN(m - ii, b_size);                                                       \
            for (size_t jj = 0; jj < n; jj += b_size) {                                                 \
                size_t jb_size = MIN(n - jj, b_size);                                                   \
                for (size_t pp = 0; pp < k; pp += b_size) {                                             \
                    size_t pb_size = MIN(k - pp, b_size);                                               \
                    for (size_t lp0 = lp0##lp0; lp0 < lp0##lp0 + lp0##b_size; lp0++) {                  \
                        for (size_t lp1 = lp1##lp1; lp1 < lp1##lp1 + lp1##b_size; lp1++) {              \
                            for (size_t lp2 = lp2##lp2; lp2 < lp2##lp2 + lp2##b_size; lp2++) {          \
                                CELL(result, i, j) += CELL(a, i, p) * CELL(b, p, j);                    \
                            }                                                                           \
                        }                                                                               \
                    }                                                                                   \
                }                                                                                       \
            }                                                                                           \
        }                                                                                               \
                                                                                                        \
        printf("%ld\n", clock() - measurement_begin);                                                   \
                                                                                                        \
        *out_result = result;                                                                           \
        return 0;                                                                                       \
    }

#define DEFINE_MATMUL(lp0, lp1, lp2)                                                             \
    int matmul_##lp0##lp1##lp2(matrix_t *a, matrix_t *b, matrix_t **out_result, size_t b_size) { \
        if (b_size == 0) {                                                                       \
            return plain_matmul_##lp0##lp1##lp2(a, b, out_result);                               \
        }                                                                                        \
        return block_matmul_##lp0##lp1##lp2(a, b, out_result, b_size);                           \
    }

#define FOREACH_ORDER(macro) \
    macro(i, j, p)           \
    macro(i, p, j)           \
    macro(j, i, p)           \
    macro(j, p, i)           \
    macro(p, i, j)           \
    macro(p, j, i)

FOREACH_ORDER(DEFINE_PLAIN_MATMUL)
FOREACH_ORDER(DEFINE_BLOCK_MATMUL)
FOREACH_ORDER(DEFINE_MATMUL)

// ./prog.out a_matrix b_matrix out_matrix order [block_size]
// ./prog.out m1.csv   m2.csv   /dev/null  jip
// ./prog.out m1.csv   m2.csv   m3.csv     pij   2
int main(int argc, char **argv) {
    matrix_t *a = NULL;
    matrix_t *b = NULL;
    matrix_t *c = NULL;
    matmul_t *matmul = NULL;
    FILE *out = NULL;
    size_t b_size = 0;
    int ret = 1;

    if (argc != 5 && argc != 6) {
        goto cleanup;
    }
    const char *a_path = argv[1];
    const char *b_path = argv[2];
    const char *c_path = argv[3];
    const char *order = argv[4];

    if (read_matrix(a_path, &a)) {
        goto cleanup;
    }
    if (read_matrix(b_path, &b)) {
        goto cleanup;
    }

    out = fopen(c_path, "w");
    if (!out) {
        goto cleanup;
    }

#define SELECT_MATMUL(lp0, lp1, lp2)      \
    if (!strcmp(order, #lp0 #lp1 #lp2)) { \
        matmul = matmul_##lp0##lp1##lp2;  \
    }

    FOREACH_ORDER(SELECT_MATMUL)

#undef SELECT_MATMUL

    if (!matmul) {
        goto cleanup;
    }

    if (argc == 6) {
        if (sscanf(argv[5], "%zu", &b_size) != 1) {
            goto cleanup;
        }
    }

    printf("%ld\n", CLOCKS_PER_SEC);

    if (matmul(a, b, &c, b_size)) {
        goto cleanup;
    }
    write_matrix(c, out);
    ret = 0;

cleanup:
    free(a);
    free(b);
    free(c);
    if (out) {
        fclose(out);
    }
    return ret;
}
```