matrix_threads.c --

A master thread spawns separate child threads to compute each element in the resulting array. Each of the child threads is passed a pointer to a structure that contains the element indices and pointers to starting and resulting arrays.

The master thread joins to each thread, prints out the result and exits.

#include <stdio.h>
#include <pthread.h>

#define ARRAY_SIZE 10

typedef int matrix_t[ARRAY_SIZE][ARRAY_SIZE];

typedef struct {
    int id;
    int size;
    int Arow;
    int Bcol;
    matrix_t *MA, *MB, *MC;
} package_t;

matrix_t MA, MB, MC;
/ * Routine to multiply a row by a column and place element in * resulting matrix. */
void mult(int size,
    int row,
    int column,
    matrix_t MA,
    matrix_t MB,
    matrix_t MC)
{
    int position;

    MC[row][column] = 0;
    for(position = 0; position < size; position++) {
        MC[row][column] = MC[row][column] +
        ( MA[row][position] * MB[position][column] );
    }
}
/ * Routine to start off a worker thread. *
 */

void *mult_worker(void *arg)
{
    package_t *p=(package_t *)arg;

    printf("MATRIX THREAD %d: processing A row %d, B col %d\n", p->id, p->Arow, p->Bcol);

    mult(p->size, p->Arow, p->Bcol, *(p->MA), *(p->MB), *(p->MC));

    free(p);

    printf("MATRIX THREAD %d: complete\n", p->id);

    return(NULL);
}
/ * Main(): allocates matrix, assigns values, then  
 * creates threads to process rows and columns.  
 */
extern int
main(int argc, char **argv)
{
    int       size, row, column, num_threads, i;
    pthread_t *threads;       /* threads holds the thread ids of all  
        threads created, so that the  
        main thread can join with them. */
    package_t *p;             /* argument list to pass to each thread. */

    /* Currently size hardwired to ARRAY_SIZE size */
    size = ARRAY_SIZE;

    /* one thread will be created for each element of the matrix. */
    threads = (pthread_t *)malloc(size*size*sizeof(pthread_t));

    /* Fill in matrix values, currently values are hardwired */
    for (row = 0; row < size; row++) {
        for (column = 0; column < size; column++) {
            MA[row][column] = 1;
        }
    }
    for (row = 0; row < size; row++) {
        for (column = 0; column < size; column++) {
            MB[row][column] = row + column + 1;
        }
    }
    printf("MATRIX MAIN THREAD: The A array is is;\n");
    for(row = 0; row < size; row++) {
        for (column = 0; column < size; column++) {
            printf("%5d ",MA[row][column]);
        }
        printf("\n");
    }
    printf("MATRIX MAIN THREAD: The B array is is;\n");
    for(row = 0; row < size; row++) {
        for (column = 0; column < size; column++) {
            printf("%5d ",MB[row][column]);
        }
        printf("\n");
    }
/* Process Matrix, by row, column, Create a thread to process each element in the resulting matrix*/

num_threads = 0;
for (row = 0; row < size; row++) {
    for (column = 0; column < size; column++) {
        p = (package_t *)malloc(sizeof(package_t));
        p->id = num_threads;
        p->size = size;
        p->Arow = row;
        p->Bcol = column;
        (p->MA) = &MA;
        (p->MB) = &MB;
        (p->MC) = &MC;

        pthread_create(&threads[num_threads],
                       NULL,
                       mult_worker,
                       (void *) p);

        printf("MATRIX MAIN THREAD: thread %d created\n", num_threads);
        num_threads++;
    }
}

/* Synchronize on the completion of the element in each thread. */
for (i = 0; i < (size*size); i++) {
    pthread_join(threads[i], NULL);
    printf("MATRIX MAIN THREAD: child %d has joined\n", i);
}

/* Print results */
printf("MATRIX MAIN THREAD: The resulting matrix C is;\n");
for (row = 0; row < size; row++) {
    for (column = 0; column < size; column++) {
        printf("%5d ",MC[row][column]);
    }
    printf("\n");
}
return 0;