System Programming homework 4

B09902133

January 2022

1 Execution time in the large test case when the number of thread is 2 or 20

Result:

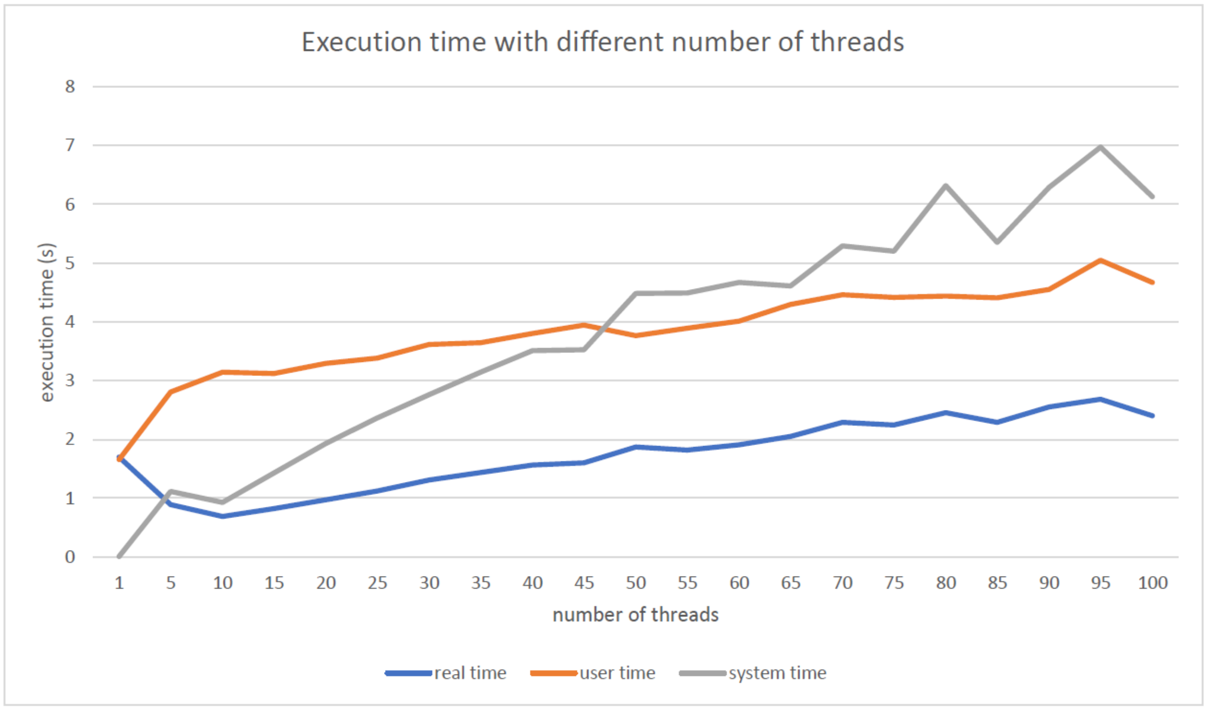
- Thread number: 2
  Real time: 2.418 s
  User time: 2.151 s
  System time: 0.414s

- Thread number: 20
  Real time: 0.956 s
  User time: 3.324s
  System time: 1.546s

Since there are multiple threads being executed simultaneously, user time is greater than real time when the thread number is 20. The result also showed that real time is smaller when we use more threads, which might be the effect of parallel work. However, the user time is larger when we use more threads, which might result from the cost of maintaining thread tables and switching between threads.

2 Execution time analysis with different number of threads

For each number of threads, I rum the program for three times and use the average real time / user time / system time to make the graph. All of the tests were done in linux2 workstation between 17:00 to 18:00 on 1/7.



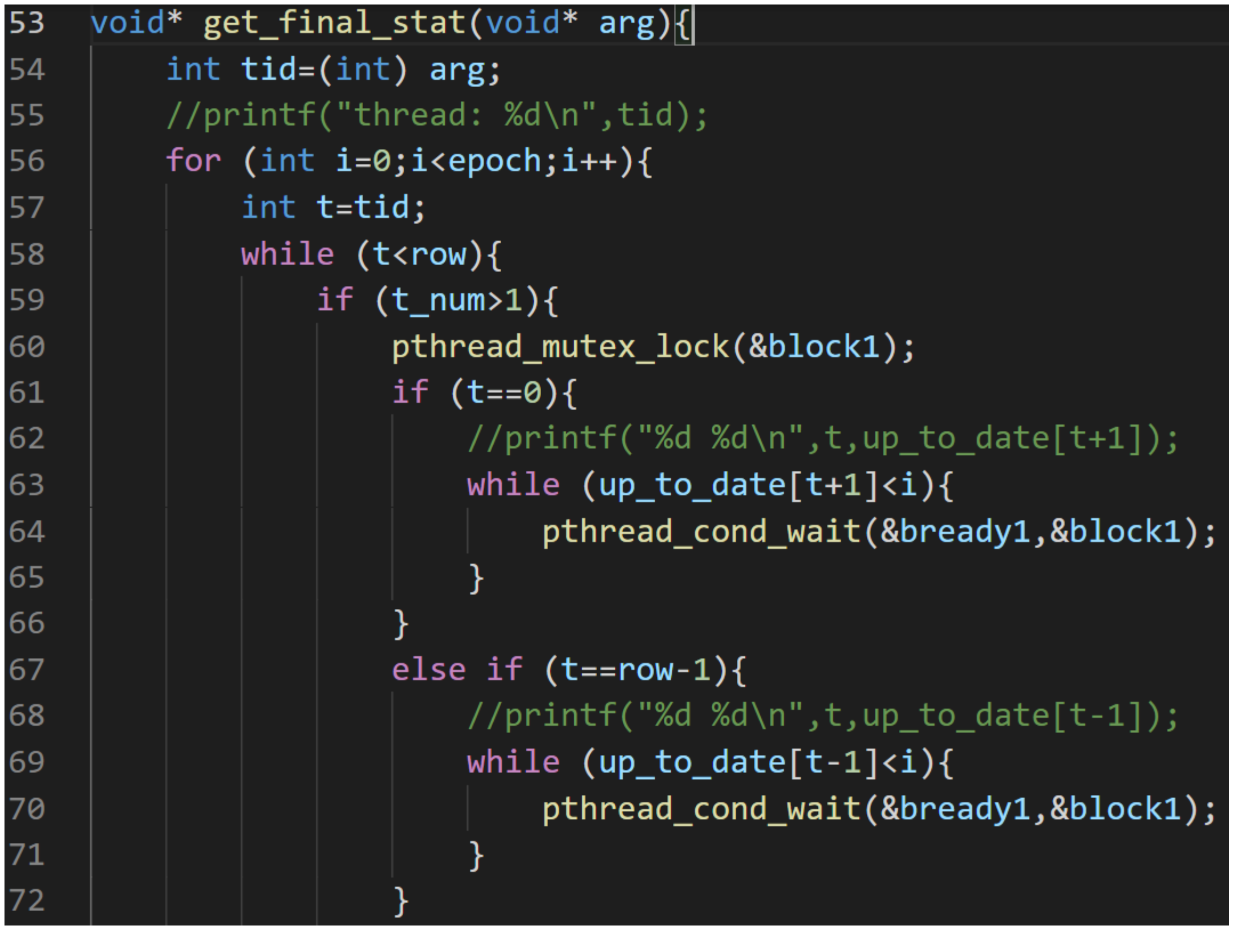
1. Real time:
   From the chart above, we can see that the real time decreases when as the number of threads increases when the number of threads is less than 15. This may be caused by the effect of parallel work, where the problem is partitioned into multiple parts and solved simultaneously. However, the real time increases as the number of threads increases when the number of threads is more than 15. This may be caused by the limited number of processors. Though the number of threads keep increasing, those threads may not be executed parallel actually due to the limited number of processors. However, it will cost more time to switch between different threads.

2. User time:
   As the char shows, the user time increases gradually when the number of threads increases. This may be caused by the increasing cost of maintaining variable locks and thread tables.

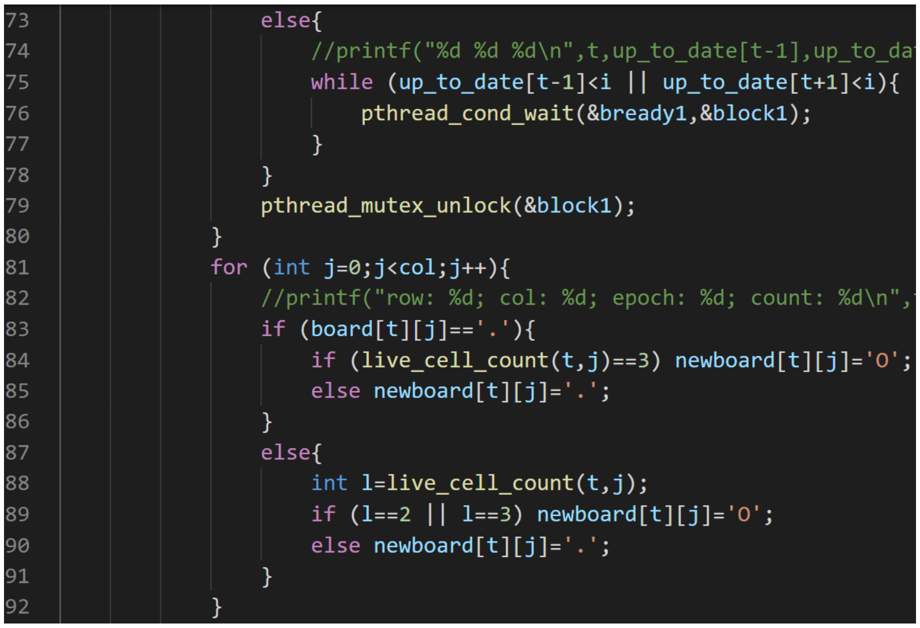
3. system time:
   As the chart shows, the system time increases when the number of threads increases. This may be caused by the increasing cost of system call related to thread creation and thread context switch.

3 The critical part of the program

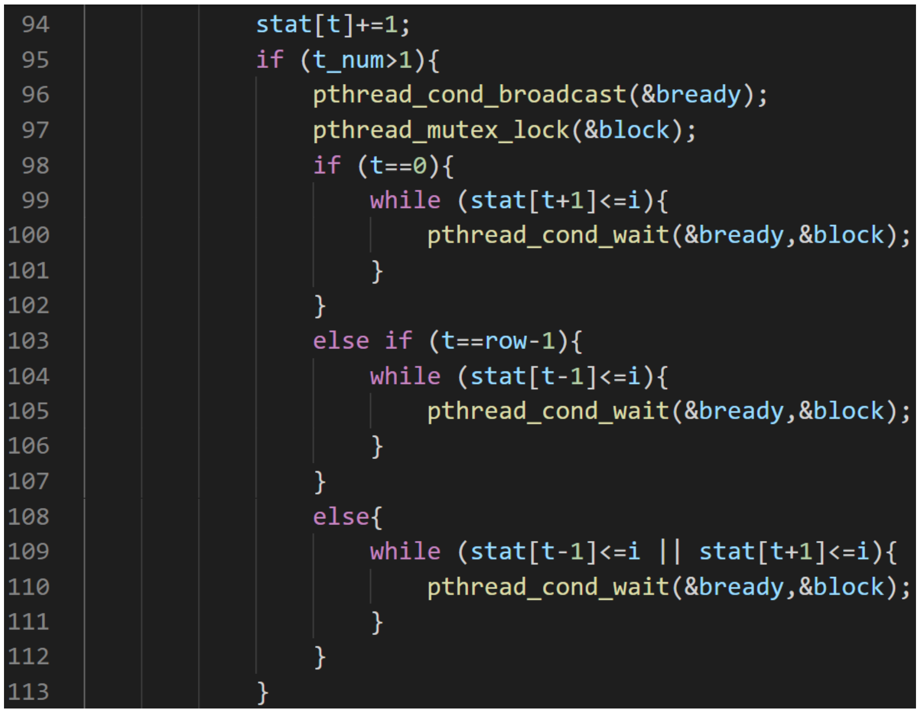
board is the 2-D array representing status of board before each epoch. *newboard* is the 2-D array representing status of board after the operation of the current epoch. Each thread is responsible for updating a row, and it will continue to update another row after finishing current row. (If the number of threads is larger than the number of rows, than transpose the board) Before each thread tries to get the newboard status, it will first calls pthread_cond_wait() to wait for the adjacent rows being up to date. Before each thread tries to update the board, it will call pthread_cond_wait() to assure that threads responsible for adjacent rows have retrieved the needed information about this row and thus this row can be updated.
The following is the critical part of my program. (the start routine of each spawned thread)



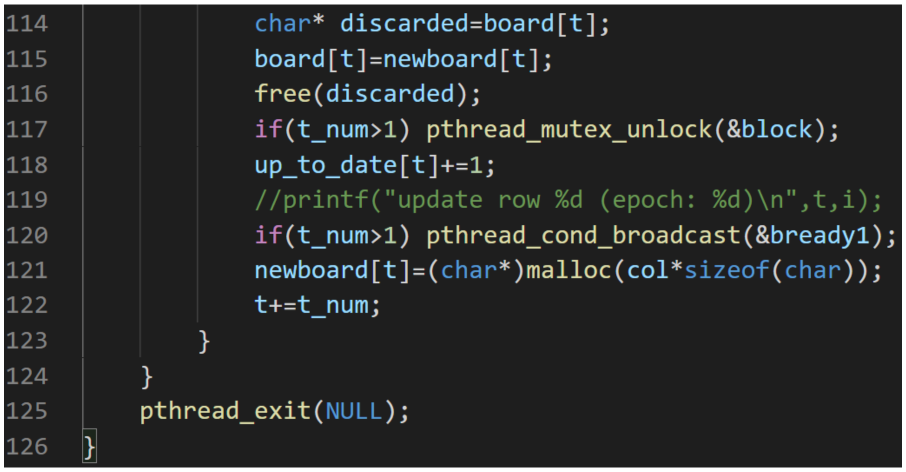
```c
void* get_final_stat(void* arg){
    int tid=(int) arg;
    //printf("thread: %d\n",tid);
    for (int i=0;i<epoch;i++){
        int t=tid;
        while (t<row){
            if (t_num>1){
                pthread_mutex_lock(&block1);
                if (t==0){
                    //printf("%d %d\n",t,up_to_date[t+1]);
                    while (up_to_date[t+1]<i){
                        pthread_cond_wait(&bready1,&block1);
                    }
                }
                else if (t==row-1){
                    //printf("%d %d\n",t,up_to_date[t-1]);
                    while (up_to_date[t-1]<i){
                        pthread_cond_wait(&bready1,&block1);
                    }
                }
```



```c
                else{
                    //printf("%d %d %d\n",t,up_to_date[t-1],up_to_da
                    while (up_to_date[t-1]<i || up_to_date[t+1]<i){
                        pthread_cond_wait(&bready1,&block1);
                    }
                }
                pthread_mutex_unlock(&block1);
            }
            for (int j=0;j<col;j++){
                //printf("row: %d; col: %d; epoch: %d; count: %d\n",
                if (board[t][j]=='.'){
                    if (live_cell_count(t,j)==3) newboard[t][j]='O';
                    else newboard[t][j]='.';
                }
                else{
                    int l=live_cell_count(t,j);
                    if (l==2 || l==3) newboard[t][j]='O';
                    else newboard[t][j]='.';
                }
            }
```



```c
            stat[t]+=1;
            if (t_num>1){
                pthread_cond_broadcast(&bready);
                pthread_mutex_lock(&block);
                if (t==0){
                    while (stat[t+1]<=i){
                        pthread_cond_wait(&bready,&block);
                    }
                }
                else if (t==row-1){
                    while (stat[t-1]<=i){
                        pthread_cond_wait(&bready,&block);
                    }
                }
                else{
                    while (stat[t-1]<=i || stat[t+1]<=i){
                        pthread_cond_wait(&bready,&block);
                    }
                }
            }
```



```c
            char* discarded=board[t];
            board[t]=newboard[t];
            free(discarded);
            if(t_num>1) pthread_mutex_unlock(&block);
            up_to_date[t]+=1;
            //printf("update row %d (epoch: %d)\n",t,i);
            if(t_num>1) pthread_cond_broadcast(&bready1);
            newboard[t]=(char*)malloc(col*sizeof(char));
            t+=t_num;
        }
    }
    pthread_exit(NULL);
}
```