Project 1.

1. GTThreads package

The GTThreads package consists of the following components:

- ***gt_bitops*** - describes the functions for assigning and resetting the bitmask, as well as the function for determining the lowest bit set (needed to define the highest priority for uthread);
- ***gt_signal*** - describes the functions for sending signals (SIGVTALRM, etc.), which are used to interrupt the execution of the current thread after a certain time interval;
- ***gt_a spinlock*** - describes the function of the spinlock, which allows threads to "wait" while the current thread is executing;
- ***gt_tailq*** - describes various algorithms for adding (at the head of the queue, at the end of the queue, etc.) and removing uthread from kthread ;
- ***gt_pq*** - describes functions for finding a suitable uthread to add to the runqueue. It also includes queue and priority structures;
- ***gt_kthread*** - describes the functions of kthread's operations (such as creation, initialization, etc.) and also gtthread application's operations over kthreads and uthreads (starting, initialization and termination);
- ***gt_uthread*** - describes the functions of uthread's operations (such as creation, initialization, etc.) ;
- ***gt_matrix*** - the main program for creating and multiplying matrices :
    - gtthread_app_init - initializes, allocates memory and creates kthread ;
    - init_matrices - initializes and creates 3 matrices;
    - uthread_create (& utids [inx], uthread _ mulmat , uarg , uarg -> gid) - allocates memory and creates a new uthread , then adds it to the active runqueue for the selected kthread;
    - gtthread_app_exit - while there are active uthreads, it executes the uthread_schedule (& sched_find_best_uthread) function . The function's main task is to move the uthread between runqueues and shift active and expires runqueues.

1.1 Implementation of the O(1) priority scheduler:

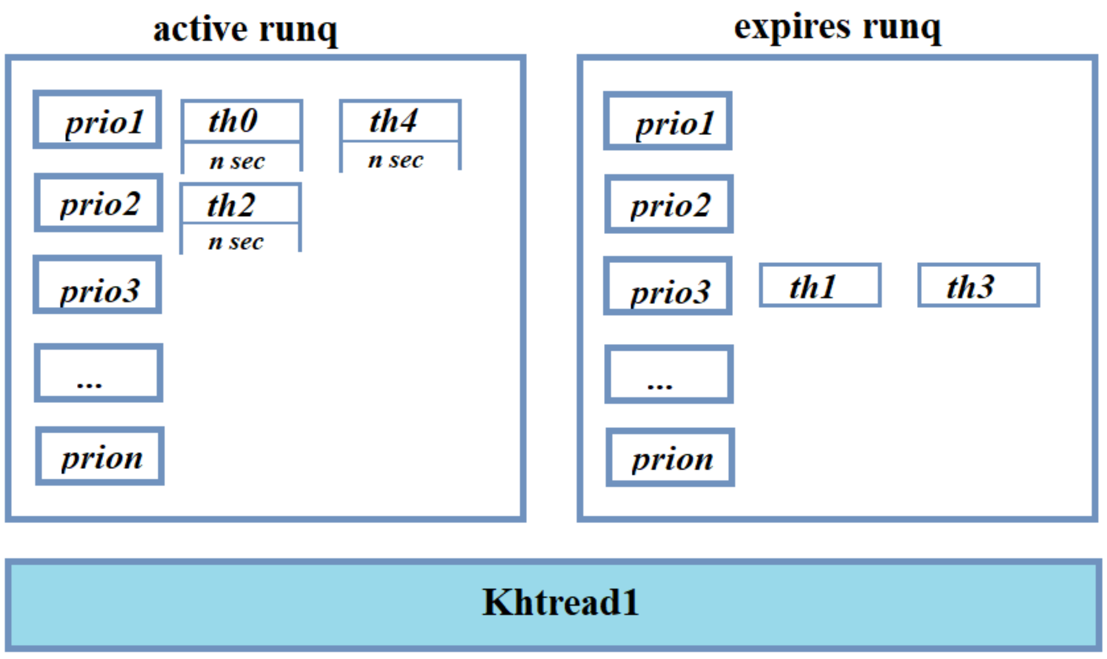


Figure 1. The algorithm of O(1)

Initially, the program creates kthread on the first logical processor using the "kthread_init" function in "gtthread_app_init". All subsequent kthreads (depending on the number of logical processors) will be its "clones" according to the "kthread_create" function. Each kthread has two runqueues active and expires (Figure 1).

At the time of creation, a uthread is assigned to a specific kthread using the "ksched_find_target" function. Then the uthread is added to the kthread's active runqueue by the priority (the least bit set corresponds to the highest priority).

Following the function "uthread _ schedule (& sched _ find _ best _ uthread)", each uthread in the "*Running*" state will be executed until it receives a signal "SIGVTALRM" (timeslice is set in gt _ signal.h). At this time, all other uthreads will receive the SIGUSR1 signal. After receiving a signal, if the uthread is not finished, its state will be changed to "*Runnable*", and it will be transferred to the expires queue. If the work is completed, then the uthread's state will become "*Done*", and it will be moved to the zombie runqueue with decreasing the value of "kthread_cur_uthreads". The "uthread_context_func" function is responsible for changing the uthread state.

If there is no uthread left in the active queue ("uthread_tot" = 0), the active and expires runqueues will be swapped. The "sched_find_best_uthread" function is responsible for changing runqueues. After that, the execution of uthread will start again. The algorithm will repeat until all uthreads have finished executing. A simplified algorithm is shown below (Figure 2).

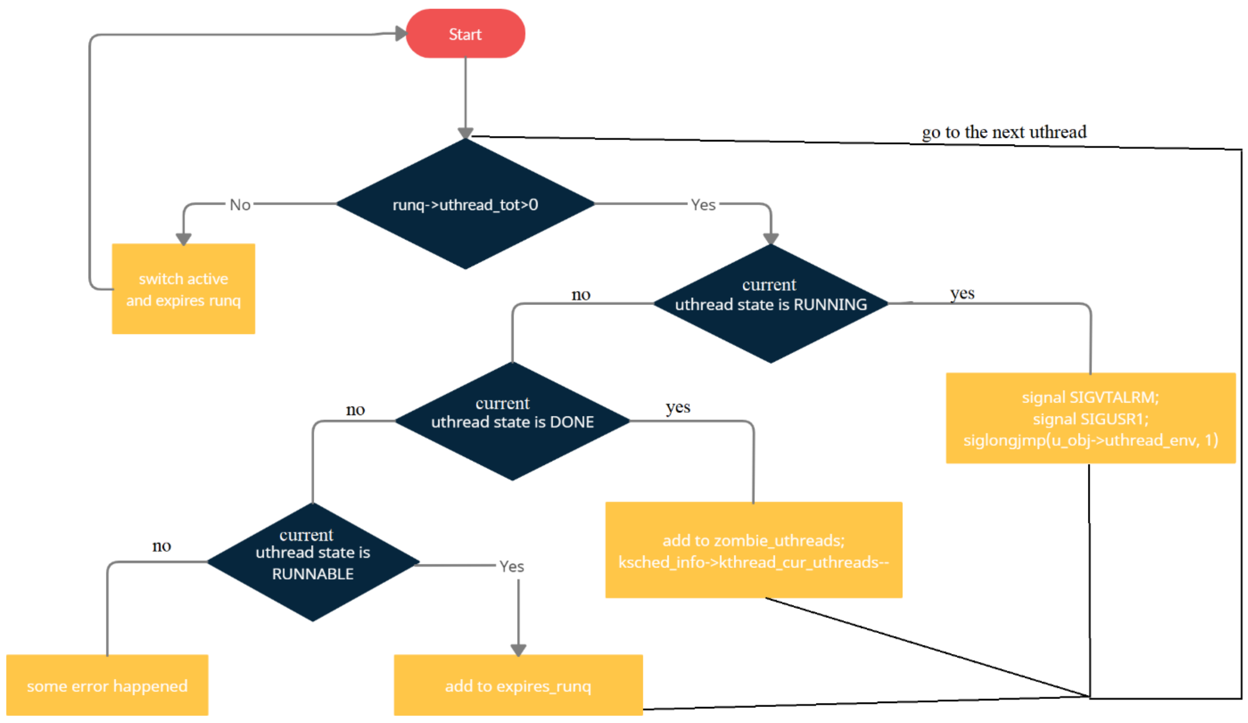

Start
go to the next uthread
runq->uthread_tot>0
No
Yes
switch active and expires runq
current uthread state is RUNNING
no
yes
signal SIGVTALRM;
signal SIGUSR1;
siglongjmp(u_obj->uthread_env, 1)
current uthread state is DONE
no
yes
add to zombie_uthreads;
ksched_info->kthread_cur_uthreads--
current uthread state is RUNNABLE
no
Yes
some error happened
add to expires_runq

Figure 2. Also the algorithm of O(1)

2. The main ideas of Credit scheduler

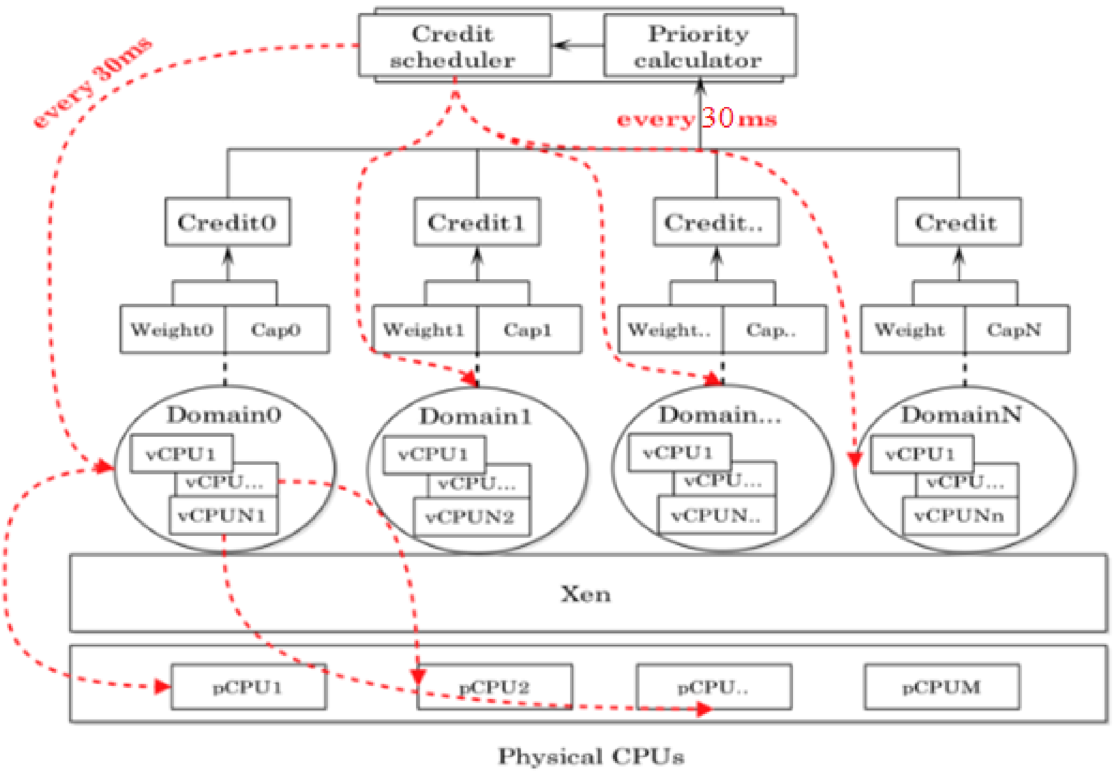


Figure 3. Algorithm of credit scheduler[1]

Credit scheduler is a fair share virtual CPU scheduler. The scheduler awards credit to each vCPU periodically and charges each vCPU to run on the pCPU[1].

Each **credit** contains two key **parameters**:

- *Weight* is proportional to vCPU value. That is, the higher a vCPU's weight, the more pCPU it can use;
- *Cap* is expressed as a percentage of the maximum number of pCPUs that a domain can utilize[1].

Each vCPU can have two **priorities**:

- *"Over"*, if the number of credits is less than 0;
- *"Under"*, if the number of credits is more than 0.

Time in Credit scheduler:

- *Time* slice (scheduling quantum) - uses 30 ms by default, after which it interrupts the vCPU and recalculates its priority;
- *Context-switch rate limit* - use 1ms by default. It is defining the minimum number of microseconds a vCPU would be allowed to run uninterrupted before being context switched[1].

Algorithm.

Each physical processor manages a queue of the vCPU. vCPUs are sorted based on priority ("Over" or "Under"). After a certain amount of time (30 ms), the vCPU will be interrupted, and the execution of the next vCPU will start. While the vCPU's priority is "Under", it will be placed at the end of the queue. The vCPU with priority "Over" will be able to run before any that are out of credit. Credit priorities are recounted every 30 ms [2].

Thus, we can conclude that threads with more credits will have more time to execute.

3. Implementation of the Credit based scheduler

3.1 Algorithm

Comparing the O(1) and the Credit schedulers' work, we can conclude that their principles are similar. In the Credit scheduler a kthread has a runqueue of runnable uthreads (for this project, we can assume that each kthread has 2 runqueues Under and Over) (Figure 4). A uthread will work a specific time before being interrupted (as in the O (1) scheduler). However, in a Credit scheduler, "more important" threads with more credits must execute longer before being preempted by another thread.

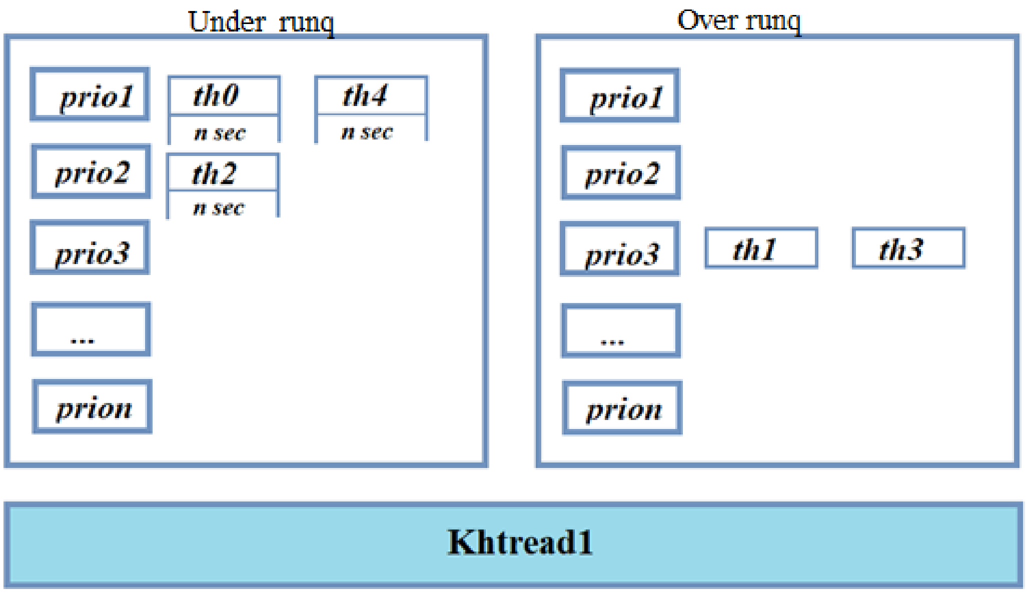
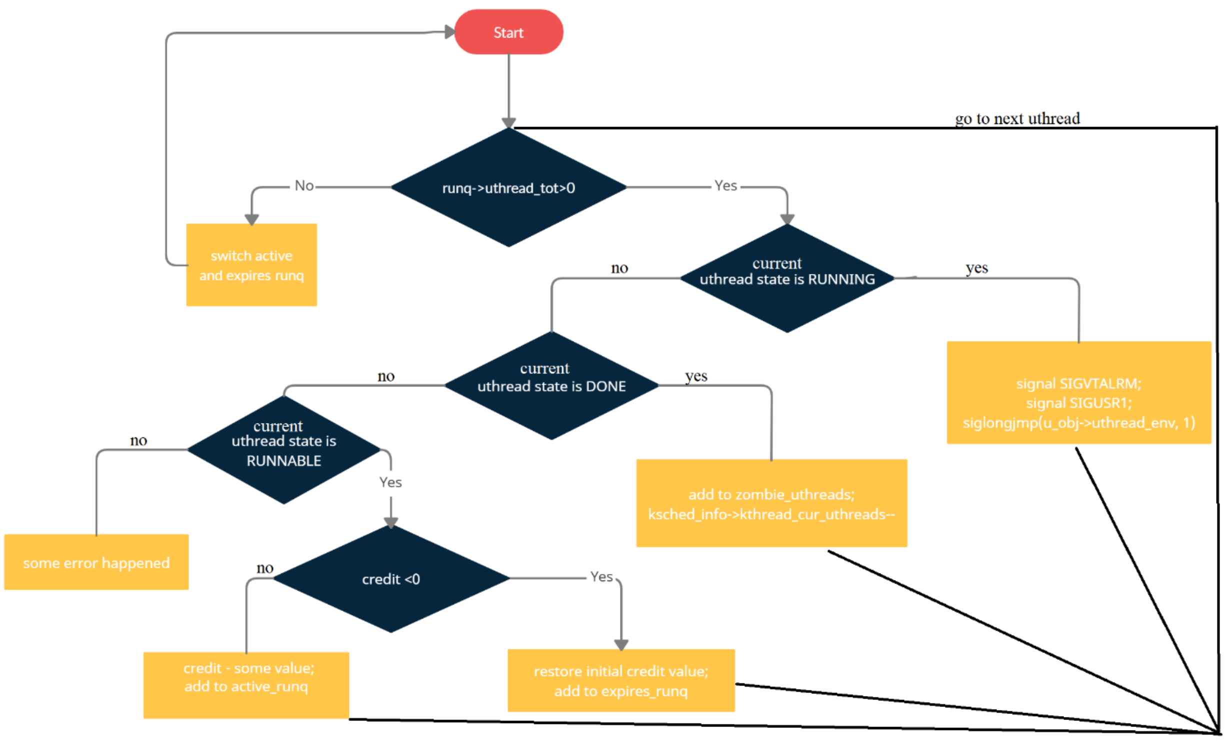


Figure 4. The credit scheduler algorithm

We can implement it that way: after each performance, the uthread will lose a certain number of credits. Before the next execution starts, the number of the thread's credits must be checked. If the number of credits is greater than 0, then the uthread is added to the end of Under queue (active); if the number of credits is lower than 0, then the uthread will be transferred to Over queue (expires). However, we should keep in mind, that when uthreads get into Over queue, they should get their initial credits back. When there are no active uthreads left in Under queue, then queues will change as in the O (1) scheduler. The algorithm is shown below (Figure 5).

In that way the main idea of the credit scheduler will be saved. For example, in our project we can assume, that uthread loses 25 credits when executed, so a uthread with 100 credits will be executed four times in Under queue, while uthread with 25 credits will only be executed once. Only after that, they will be transferred to the Over queue.



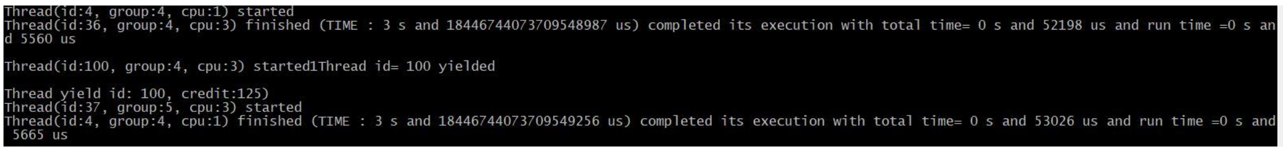
Start
go to next uthread
runq->uthread_tot>0
No
Yes
switch active and expires runq
current uthread state is RUNNING
no
yes
signal SIGVTALRM; signal SIGUSR1; siglongjmp(u_obj->uthread_env, 1)
current uthread state is DONE
no
yes
current uthread state is RUNNABLE
no
Yes
add to zombie_uthreads; ksched_info->kthread_cur_uthreads--
some error happened
credit <0
no
Yes
credit - some value; add to active_runq
restore initial credit value; add to expires_runq

Figure 5. Also the credit scheduler algorithm

3.2 gt_yield

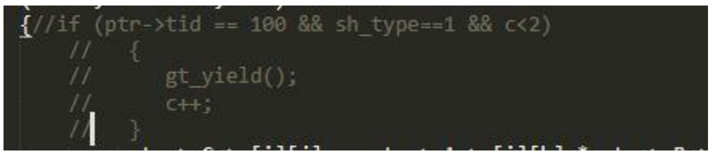
In this project the gt_yield function has been added to complete the execution voluntarily. When it is called, the function changes the corresponding flag(gt_yield_flag) from 0 to 1 (added to uthread_struct). The “uthread_schedule” function checks the value of this flag, and if the flag is equal to one, then the credit value of the yielding uthread is changed (in our case, it adds 25 credits extra), and the uthread is moved to the expires runqueue. In real life, it is usually used with I/O operations.

During testing, the “gt_yield” function was called for an arbitrary uthread. The result is shown in Figure 6. The function is disabled by default. To active it, the below code should be uncommented (Figure 7).



```
Thread(id:4, group:4, cpu:1) started
Thread(id:36, group:4, cpu:3) finished (TIME : 3 s and 18446744073709548987 us) completed its execution with total time= 0 s and 52198 us and run time =0 s an
d 5560 us

Thread(id:100, group:4, cpu:3) started1Thread id= 100 yielded

Thread yield id: 100, credit:125)
Thread(id:37, group:5, cpu:3) started
Thread(id:4, group:4, cpu:1) finished (TIME : 3 s and 18446744073709549256 us) completed its execution with total time= 0 s and 53026 us and run time =0 s and
 5665 us
```

Figure 6. Gt _ yield ( )

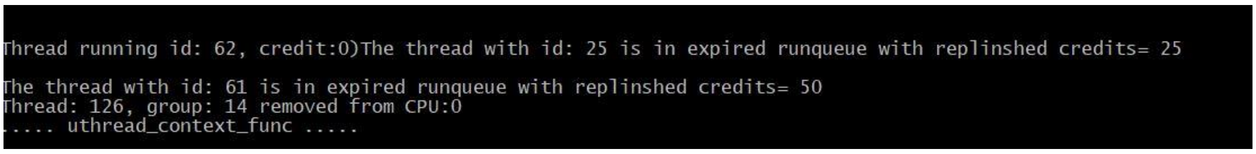


```
{//if (ptr->tid == 100 && sh_type==1 && c<2)
    //   {
    //       gt_yield();
    //       c++;
    //   }
```

Figure 7. Code of gt_yield()

3.3 Load balancing

The balancing algorithm was also added in the Credit scheduler. It is used when a certain kthread is idle, but there are still uthreads to execute. So a free kthread can intercept them. To do this, we introduce a new runqueue (kthread_runqueue_t * other_kthread_runq; runqueue_t * other_runq). If other runqueue's uthread_mask value is not zero, then a free kthread can intercept some uthreads. The result is shown below in Figure 8.



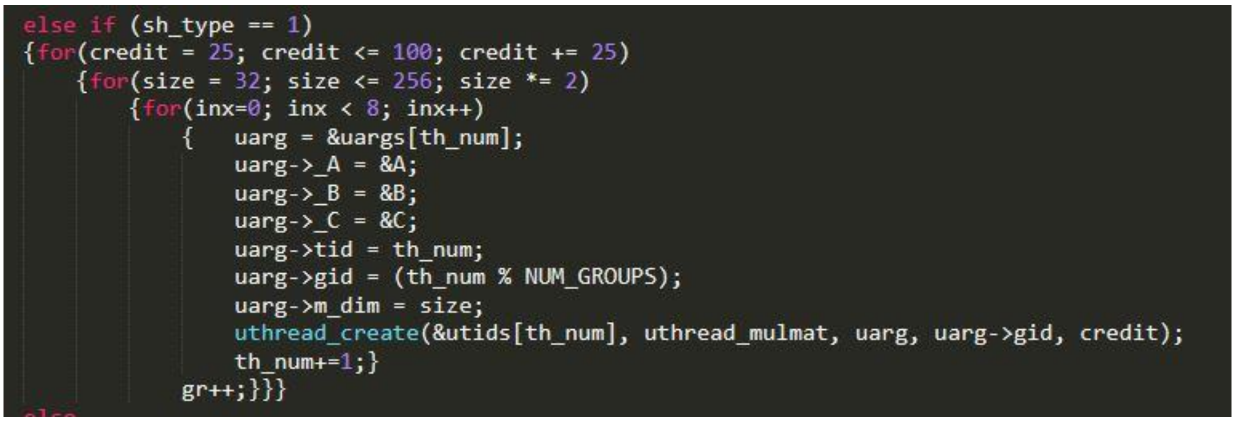
```
hread running id: 62, credit:0)The thread with id: 25 is in expired runqueue with replinshed credits= 25

he thread with id: 61 is in expired runqueue with replinshed credits= 50
hread: 126, group: 14 removed from CPU:0
.... uthread_context_func .....
```

Figure 8. Load balancing

3.4 Matrix

Each utread is executed on a matrix of its own. We create 128 uthreads and divide them into 16 groups (according to NUM_GROUPS). In each group, we have eight uthreads with a different number of credits and matrix size.

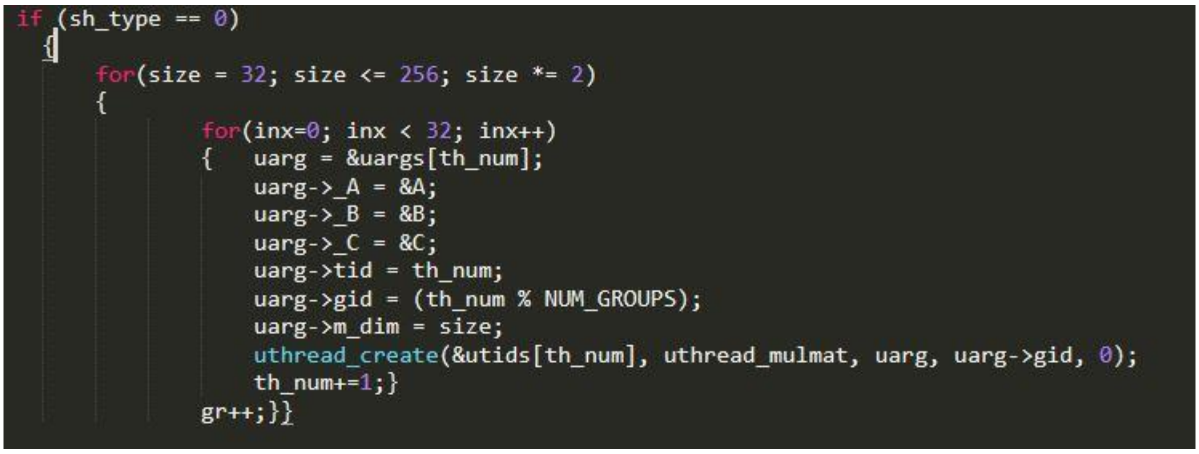
The code is provided below in Figure 9.



```
else if (sh_type == 1)
{for(credit = 25; credit <= 100; credit += 25)
    {for(size = 32; size <= 256; size *= 2)
        {for(inx=0; inx < 8; inx++)
            {   uarg = &uargs[th_num];
                uarg->_A = &A;
                uarg->_B = &B;
                uarg->_C = &C;
                uarg->tid = th_num;
                uarg->gid = (th_num % NUM_GROUPS);
                uarg->m_dim = size;
                uthread_create(&utids[th_num], uthread_mulmat, uarg, uarg->gid, credit);
                th_num+=1;}
            gr++;}}}
```

Figure 9. Matrix for the Credit scheduler

For the O(1) scheduler, we also create 128 uthreads. However, since there are no credits, there are 32 the same uthreads for each matrix size. The code is also provided below in Figure 10.



```
if (sh_type == 0)
  {

     for(size = 32; size <= 256; size *= 2)
     {
             for(inx=0; inx < 32; inx++)
             {   uarg = &uargs[th_num];
                 uarg->_A = &A;
                 uarg->_B = &B;
                 uarg->_C = &C;
                 uarg->tid = th_num;
                 uarg->gid = (th_num % NUM_GROUPS);
                 uarg->m_dim = size;
                 uthread_create(&utids[th_num], uthread_mulmat, uarg, uarg->gid, 0);
                 th_num+=1;}
             gr++;}}
```

Figure 10. Matrix for the O (1) scheduler

3.5 Time calculating

To calculate the time, we add new elements to the uthread_struct structure:

struct timeval time_start; /*time when uthread is created */
struct timeval time_end; /* time when uthread's state is done*/
struct timeval run_time; /* total time of uthread's execution*/
struct timeval run_time_t; /* uthread's executing time*/
struct timeval time_pr; /*time when uthread is prempted by another uthread*/
struct timeval time_sh; /*time when uthread adds to runq */.

Also, a new structure “uthread_time_t” is added:
struct timeval runtime; /*total time of executing*/
struct timeval totaltime; /* total time*/

The calculations:

totaltime = time_end – start_time;

run_time_t = time_pr - time_sh;

run_time = сумма всех run_time_t.

The standard deviation and the mean value of both, the individual thread run times and the total execution times, should be identified.

To find the mean value, we added the runtime and totaltime of all threads in one group and divided it by 8 (the number of threads in one group).

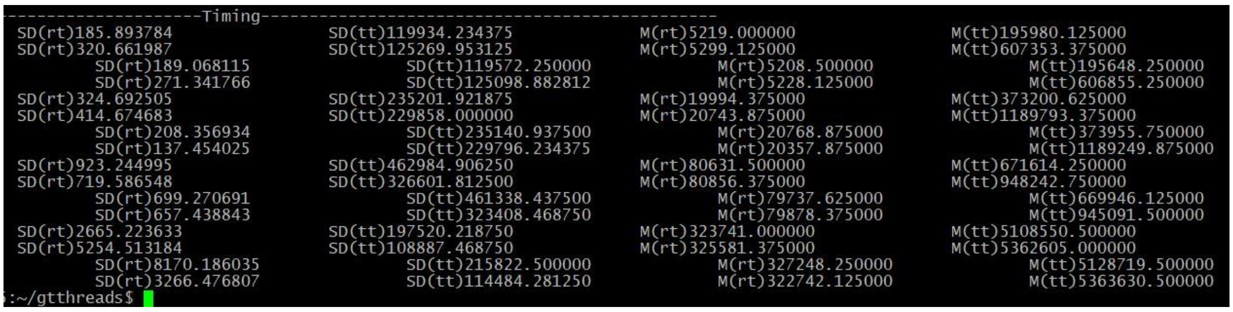
In order to find the value of the standard deviation, we used the formula (Figure 11), where m is the average value, x is the time value, N = 8:

$$\mathrm{SD} = \sqrt{\frac{\sum |x - \mu|^2}{N}}$$

Figure 11. Formula of standard deviation

4. Conclusion

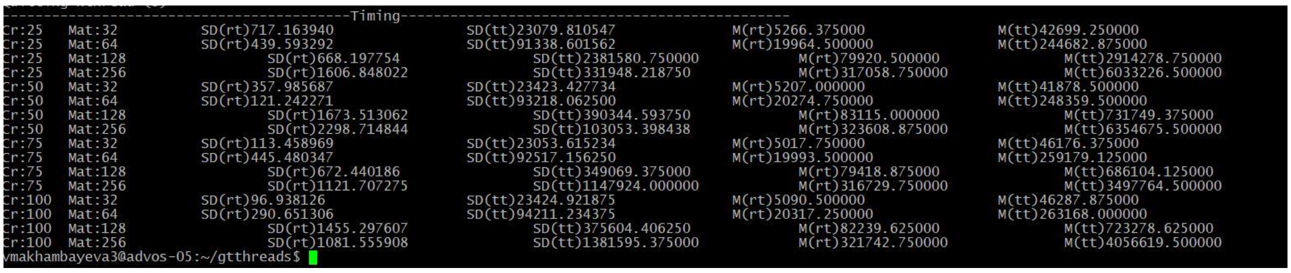
During the experiments, where the NUM_CPUS = 4 was assumed, the following results were obtained :



```
--------------------Timing---------------------------------------------
 SD(rt)185.893784                 SD(tt)119934.234375              M(rt)5219.000000                 M(tt)195980.125000
 SD(rt)320.661987                 SD(tt)125269.953125              M(rt)5299.125000                 M(tt)607353.375000
         SD(rt)189.068115                 SD(tt)119572.250000              M(rt)5208.500000                 M(tt)195648.250000
         SD(rt)271.341766                 SD(tt)125098.882812              M(rt)5228.125000                 M(tt)606855.250000
 SD(rt)324.692505                 SD(tt)235201.921875              M(rt)19994.375000                M(tt)373200.625000
 SD(rt)414.674683                 SD(tt)229858.000000              M(rt)20743.875000                M(tt)1189793.375000
         SD(rt)208.356934                 SD(tt)235140.937500              M(rt)20768.875000                M(tt)373955.750000
         SD(rt)137.454025                 SD(tt)229796.234375              M(rt)20357.875000                M(tt)1189249.875000
 SD(rt)923.244995                 SD(tt)462984.906250              M(rt)80631.500000                M(tt)671614.250000
 SD(rt)719.586548                 SD(tt)326601.812500              M(rt)80856.375000                M(tt)948242.750000
         SD(rt)699.270691                 SD(tt)461338.437500              M(rt)79737.625000                M(tt)669946.125000
         SD(rt)657.438843                 SD(tt)323408.468750              M(rt)79878.375000                M(tt)945091.500000
 SD(rt)2665.223633                SD(tt)197520.218750              M(rt)323741.000000               M(tt)5108550.500000
 SD(rt)5254.513184                SD(tt)108887.468750              M(rt)325581.375000               M(tt)5362605.000000
         SD(rt)8170.186035                SD(tt)215822.500000              M(rt)327248.250000               M(tt)5128719.500000
         SD(rt)3266.476807                SD(tt)114484.281250              M(rt)322742.125000               M(tt)5363630.500000
:~/gtthreads$
```

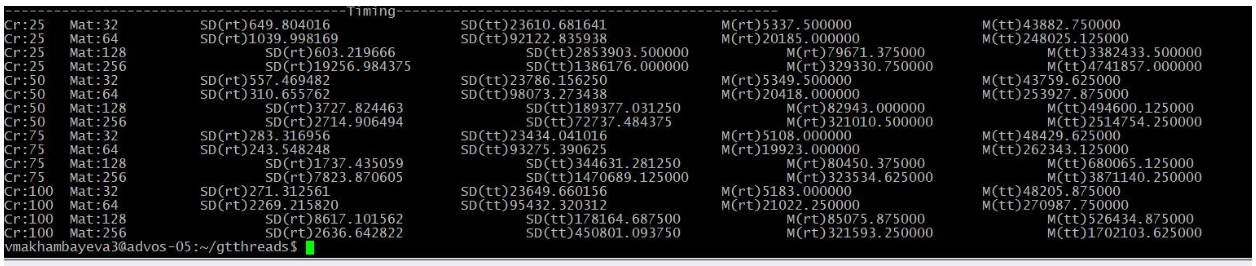
O(1) scheduler ts=100ms

For the O1 scheduler, just kept in mind, that every 4 groups have a matrix of the same size (1-4 line: 32; 5-8 line: 64; 9-12 line: 128; 13-16 line: 256). Because printing operation was done for credit scheduler, the first two columns are not valid for O(1).



```
----------------------------------------Timing----------------------------------------
Cr:25    Mat:32          SD(rt)717.163940                 SD(tt)23079.810547               M(rt)5266.375000                 M(tt)42699.250000
Cr:25    Mat:64          SD(rt)439.593292                 SD(tt)91338.601562               M(rt)19964.500000                M(tt)244682.875000
Cr:25    Mat:128                 SD(rt)668.197754                 SD(tt)2381580.750000             M(rt)79920.500000                M(tt)2914278.750000
Cr:25    Mat:256                 SD(rt)1606.848022                SD(tt)331948.218750              M(rt)317058.750000               M(tt)6033226.500000
Cr:50    Mat:32          SD(rt)357.985687                 SD(tt)23423.427734               M(rt)5207.000000                 M(tt)41878.500000
Cr:50    Mat:64          SD(rt)121.242271                 SD(tt)93218.062500               M(rt)20274.750000                M(tt)248359.500000
Cr:50    Mat:128                 SD(rt)1673.513062                SD(tt)390344.593750              M(rt)83115.000000                M(tt)731749.375000
Cr:50    Mat:256                 SD(rt)2298.714844                SD(tt)103053.398438              M(rt)323608.875000               M(tt)6354675.500000
Cr:75    Mat:32          SD(rt)113.458969                 SD(tt)23053.615234               M(rt)5017.750000                 M(tt)46176.375000
Cr:75    Mat:64          SD(rt)445.480347                 SD(tt)92517.156250               M(rt)19993.500000                M(tt)259179.125000
Cr:75    Mat:128                 SD(rt)672.440186                 SD(tt)349069.375000              M(rt)79418.875000                M(tt)686104.125000
Cr:75    Mat:256                 SD(rt)1121.707275                SD(tt)1147924.000000             M(rt)316729.750000               M(tt)3497764.500000
Cr:100   Mat:32          SD(rt)96.938126                  SD(tt)23424.921875               M(rt)5090.500000                 M(tt)46287.875000
Cr:100   Mat:64          SD(rt)290.651306                 SD(tt)94211.234375               M(rt)20317.250000                M(tt)263168.000000
Cr:100   Mat:128                 SD(rt)1455.297607                SD(tt)375604.406250              M(rt)82239.625000                M(tt)723278.625000
Cr:100   Mat:256                 SD(rt)1081.555908                SD(tt)1381595.375000             M(rt)321742.750000               M(tt)4056619.500000
vmakhambayeva3@advos-05:~/gtthreads$
```

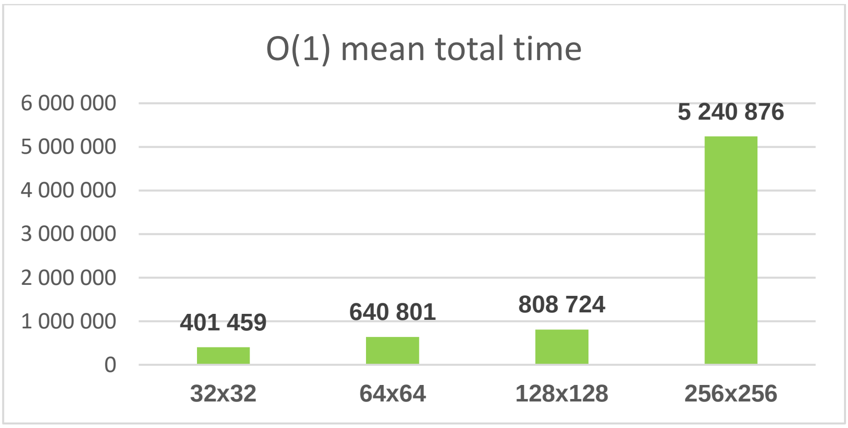
Credit scheduler ts=100ms



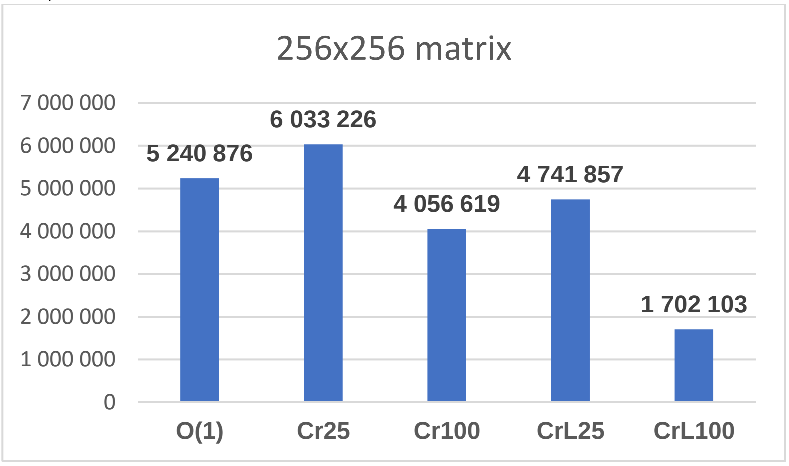
```
----------------------------------------Timing----------------------------------------
Cr:25    Mat:32          SD(rt)649.804016                 SD(tt)23610.681641               M(rt)5337.500000                 M(tt)43882.750000
Cr:25    Mat:64          SD(rt)1039.998169                SD(tt)92122.835938               M(rt)20185.000000                M(tt)248025.125000
Cr:25    Mat:128                 SD(rt)603.219666                 SD(tt)2853903.500000             M(rt)79671.375000                M(tt)3382433.500000
Cr:25    Mat:256                 SD(rt)19256.984375               SD(tt)1386176.000000             M(rt)329330.750000               M(tt)4741857.000000
Cr:50    Mat:32          SD(rt)557.469482                 SD(tt)23786.156250               M(rt)5349.500000                 M(tt)43759.625000
Cr:50    Mat:64          SD(rt)310.655762                 SD(tt)98073.273438               M(rt)20418.000000                M(tt)253927.875000
Cr:50    Mat:128                 SD(rt)3727.824463                SD(tt)189377.031250              M(rt)82943.000000                M(tt)494600.125000
Cr:50    Mat:256                 SD(rt)2714.906494                SD(tt)72737.484375               M(rt)321010.500000               M(tt)2514754.250000
Cr:75    Mat:32          SD(rt)283.316956                 SD(tt)23434.041016               M(rt)5108.000000                 M(tt)48429.625000
Cr:75    Mat:64          SD(rt)243.548248                 SD(tt)93275.390625               M(rt)19923.000000                M(tt)262343.125000
Cr:75    Mat:128                 SD(rt)1737.435059                SD(tt)344631.281250              M(rt)80450.375000                M(tt)680065.125000
Cr:75    Mat:256                 SD(rt)7823.870605                SD(tt)1470689.125000             M(rt)323534.625000               M(tt)3871140.250000
Cr:100   Mat:32          SD(rt)271.312561                 SD(tt)23649.660156               M(rt)5183.000000                 M(tt)48205.875000
Cr:100   Mat:64          SD(rt)2269.215820                SD(tt)95432.320312               M(rt)21022.250000                M(tt)270987.750000
Cr:100   Mat:128                 SD(rt)8617.101562                SD(tt)178164.687500              M(rt)85075.875000                M(tt)526434.875000
Cr:100   Mat:256                 SD(rt)2636.642822                SD(tt)450801.093750              M(rt)321593.250000               M(tt)1702103.625000
vmakhambayeva3@advos-05:~/gtthreads$
```

Credit scheduler with load balance ts=100ms

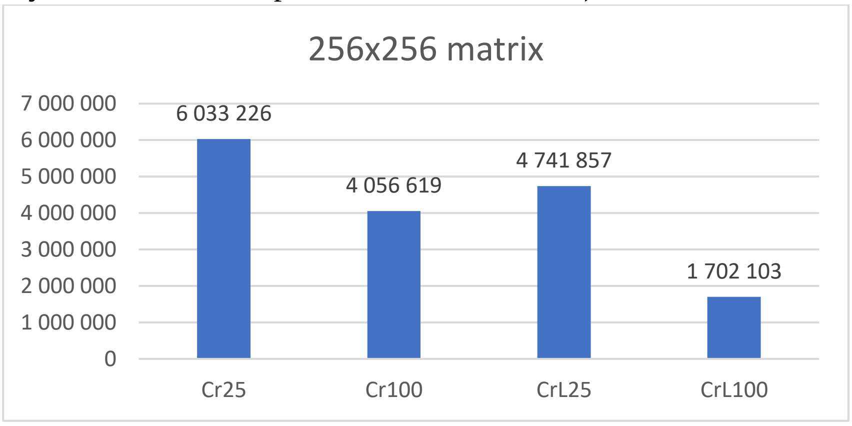
1) O(1) works as planned, run time and total time for similar processes (thread with one matrix size) are identical. You can observe a uniform increase in run time (approximately four times) depending on the matrix's size. The total execution time also increases depending on the size of the matrix;



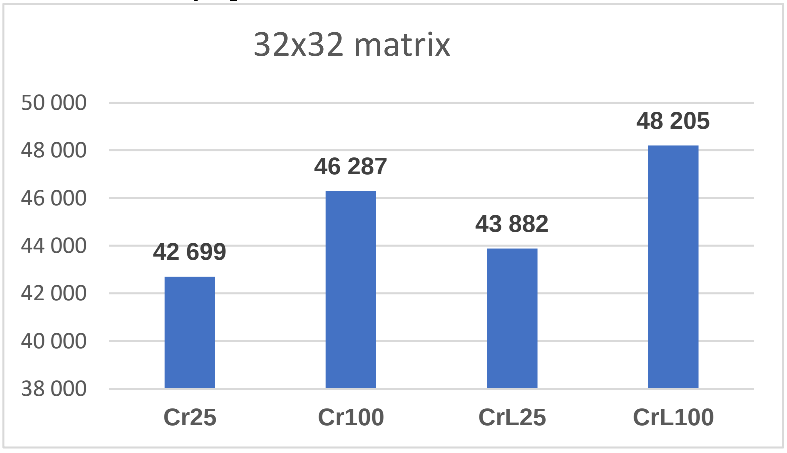
2) The number of credits affects the thread's total execution time in the Credit scheduler, as expected. The more credits, the less total execution time (it is more clearly seen on the example of 256x256 matrices);



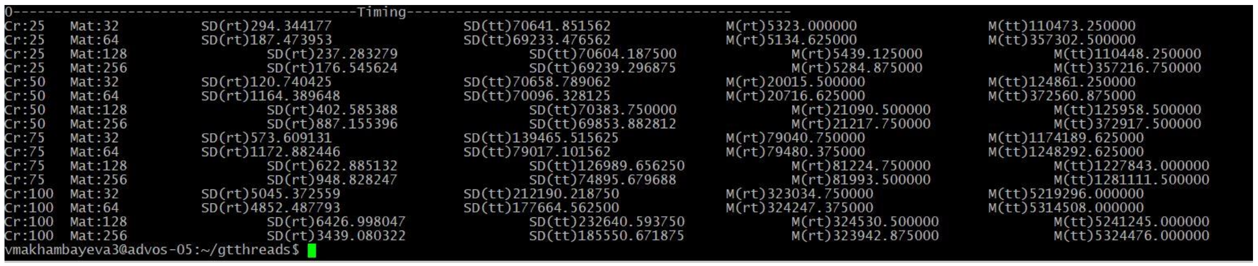
3) Credit scheduler with balancing can significantly reduce the total execution time of the uthread (it is more clearly seen on the example of 256x256 matrices);



4) For thread with small matrix computation, the results are not indicative since they manage to finish their execution before they spend all their credit.

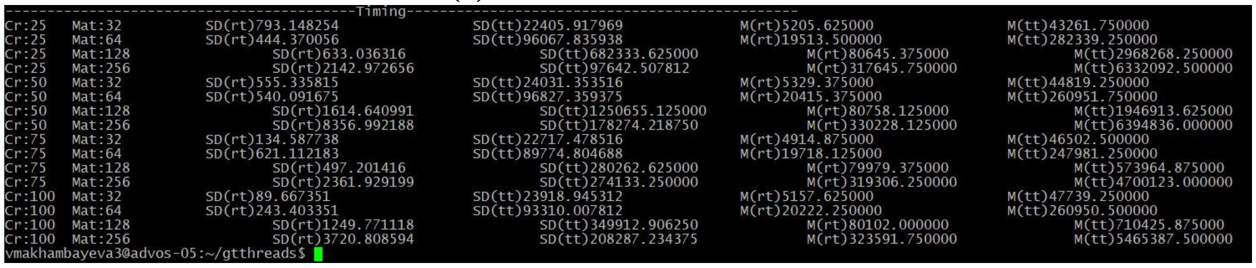


Thus, we have proved that O(1) is a fair share scheduler. However, using the credit scheduler, it is possible to execute "more important" tasks faster since they will wait less time until their turn comes. Since there were eight groups and only 4 CPUs, some experimental values may not be accurate. Similar results were obtained with ts = 30ms.



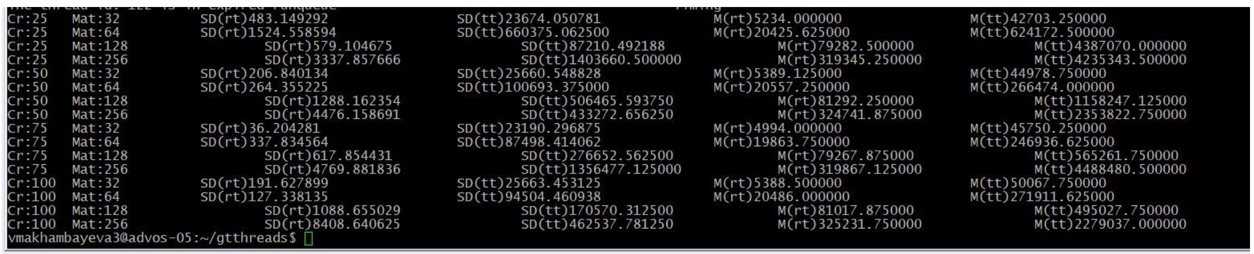
```
0-----------------------------------------Timing----------------------------------------------
Cr:25   Mat:32          SD(rt)294.344177                SD(tt)70641.851562              M(rt)5323.000000                M(tt)110473.250000
Cr:25   Mat:64          SD(rt)187.473953                SD(tt)69233.476562              M(rt)5134.625000                M(tt)357302.500000
Cr:25   Mat:128                 SD(rt)237.283279                SD(tt)70604.187500              M(rt)5439.125000                M(tt)110448.250000
Cr:25   Mat:256                 SD(rt)176.545624                SD(tt)69239.296875              M(rt)5284.875000                M(tt)357216.750000
Cr:50   Mat:32          SD(rt)120.740425                SD(tt)70658.789062              M(rt)20015.500000               M(tt)124861.250000
Cr:50   Mat:64          SD(rt)1164.389648               SD(tt)70096.328125              M(rt)20716.625000               M(tt)372560.875000
Cr:50   Mat:128                 SD(rt)402.585388                SD(tt)70383.750000              M(rt)21090.500000               M(tt)125958.500000
Cr:50   Mat:256                 SD(rt)887.155396                SD(tt)69853.882812              M(rt)21217.750000               M(tt)372917.500000
Cr:75   Mat:32          SD(rt)573.609131                SD(tt)139465.515625             M(rt)79040.750000               M(tt)1174189.625000
Cr:75   Mat:64          SD(rt)1172.882446               SD(tt)79017.101562              M(rt)79480.375000               M(tt)1248292.625000
Cr:75   Mat:128                 SD(rt)622.885132                SD(tt)126989.656250             M(rt)81224.750000               M(tt)1227843.000000
Cr:75   Mat:256                 SD(rt)948.828247                SD(tt)74895.679688              M(rt)81993.500000               M(tt)1281111.500000
Cr:100  Mat:32          SD(rt)5045.372559               SD(tt)212190.218750             M(rt)323034.750000              M(tt)5219296.000000
Cr:100  Mat:64          SD(rt)4852.487793               SD(tt)177664.562500             M(rt)324247.375000              M(tt)5314508.000000
Cr:100  Mat:128                 SD(rt)6426.998047               SD(tt)232640.593750             M(rt)324530.500000              M(tt)5241245.000000
Cr:100  Mat:256                 SD(rt)3439.080322               SD(tt)185550.671875             M(rt)323942.875000              M(tt)5324476.000000
vmakhambayeva3@advos-05:~/gtthreads$
```

O(1) scheduler ts=30ms



```
-----------------------------------------Timing----------------------------------------------
Cr:25   Mat:32          SD(rt)793.148254                SD(tt)22405.917969              M(rt)5205.625000                M(tt)43261.750000
Cr:25   Mat:64          SD(rt)444.370056                SD(tt)96067.835938              M(rt)19513.500000               M(tt)282339.250000
Cr:25   Mat:128                 SD(rt)633.036316                SD(tt)682333.625000             M(rt)80645.375000               M(tt)2968268.250000
Cr:25   Mat:256                 SD(rt)2142.972656               SD(tt)97642.507812              M(rt)317645.750000              M(tt)6332092.500000
Cr:50   Mat:32          SD(rt)555.335815                SD(tt)24031.353516              M(rt)5329.375000                M(tt)44819.250000
Cr:50   Mat:64          SD(rt)540.091675                SD(tt)96827.359375              M(rt)20415.375000               M(tt)260951.750000
Cr:50   Mat:128                 SD(rt)1614.640991               SD(tt)1250655.125000            M(rt)80758.125000               M(tt)1946913.625000
Cr:50   Mat:256                 SD(rt)8356.992188               SD(tt)178274.218750             M(rt)330228.125000              M(tt)6394836.000000
Cr:75   Mat:32          SD(rt)134.587738                SD(tt)22717.478516              M(rt)4914.875000                M(tt)46502.500000
Cr:75   Mat:64          SD(rt)621.112183                SD(tt)89774.804688              M(rt)19718.125000               M(tt)247981.250000
Cr:75   Mat:128                 SD(rt)497.201416                SD(tt)280262.625000             M(rt)79979.375000               M(tt)573964.875000
Cr:75   Mat:256                 SD(rt)2361.929199               SD(tt)274133.250000             M(rt)319306.250000              M(tt)4700123.000000
Cr:100  Mat:32          SD(rt)89.667351                 SD(tt)23918.945312              M(rt)5157.625000                M(tt)47739.250000
Cr:100  Mat:64          SD(rt)243.403351                SD(tt)93310.007812              M(rt)20222.250000               M(tt)260950.500000
Cr:100  Mat:128                 SD(rt)1249.771118               SD(tt)349912.906250             M(rt)80102.000000               M(tt)710425.875000
Cr:100  Mat:256                 SD(rt)3720.808594               SD(tt)208287.234375             M(rt)323591.750000              M(tt)5465387.500000
vmakhambayeva3@advos-05:~/gtthreads$
```

Credit scheduler ts=30ms



```
Cr:25   Mat:32          SD(rt)483.149292                SD(tt)23674.050781              M(rt)5234.000000                M(tt)42703.250000
Cr:25   Mat:64          SD(rt)1524.558594               SD(tt)660375.062500             M(rt)20425.625000               M(tt)624172.500000
Cr:25   Mat:128                 SD(rt)579.104675                SD(tt)87210.492188              M(rt)79282.500000               M(tt)4387070.000000
Cr:25   Mat:256                 SD(rt)3337.857666               SD(tt)1403660.500000            M(rt)319345.250000              M(tt)4235343.500000
Cr:50   Mat:32          SD(rt)206.840134                SD(tt)25660.548828              M(rt)5389.125000                M(tt)44978.750000
Cr:50   Mat:64          SD(rt)264.355225                SD(tt)100693.375000             M(rt)20557.250000               M(tt)266474.000000
Cr:50   Mat:128                 SD(rt)1288.162354               SD(tt)506465.593750             M(rt)81292.250000               M(tt)1158247.125000
Cr:50   Mat:256                 SD(rt)4476.158691               SD(tt)433272.656250             M(rt)324741.875000              M(tt)2353822.750000
Cr:75   Mat:32          SD(rt)36.204281                 SD(tt)23190.296875              M(rt)4994.000000                M(tt)45750.250000
Cr:75   Mat:64          SD(rt)337.834564                SD(tt)87498.414062              M(rt)19863.750000               M(tt)246936.625000
Cr:75   Mat:128                 SD(rt)617.854431                SD(tt)276652.562500             M(rt)79267.875000               M(tt)565261.750000
Cr:75   Mat:256                 SD(rt)4769.881836               SD(tt)1356477.125000            M(rt)319867.125000              M(tt)4488480.500000
Cr:100  Mat:32          SD(rt)191.627899                SD(tt)25663.453125              M(rt)5388.500000                M(tt)50067.750000
Cr:100  Mat:64          SD(rt)127.338135                SD(tt)94504.460938              M(rt)20486.000000               M(tt)271911.625000
Cr:100  Mat:128                 SD(rt)1088.655029               SD(tt)170570.312500             M(rt)81017.875000               M(tt)495027.750000
Cr:100  Mat:256                 SD(rt)8408.640625               SD(tt)462537.781250             M(rt)325231.750000              M(tt)2279037.000000
vmakhambayeva3@advos-05:~/gtthreads$
```

Credit scheduler with load balance ts=30ms

Reference:

1) "An introduction to the credit scheduler in Xen," *Welcome to xfhelen's Homepage*, 10-Feb-2017. [Online]. Available: https://xfhelen.github.io/2016/06/01/2016-06-01/. [Accessed: 1-Feb-2021].
2) "Credit Scheduler," *Xen*. [Online]. Available: https://wiki.xenproject.org/wiki/Credit_Scheduler. [Accessed: 1-Feb-2021].