

Project I : Credit Scheduling with GTThreads

Prof. Karsten Schwan

1 Goal

The goal of the project is to understand and implement a credit scheduler in a user-level threads library. To do so you must modify the provided GTThreads library. The library implements a $O(1)$ priority scheduler and a priority co-scheduler for reference.

2 Project Details

2.1 Requirements

The project minimally requires the following to receive full credit :

- Implement a Credit Scheduler for the SMP GTThreads library provided
- Implement a function for matrix multiplication, using the provided code
- Write a Makefile with the basic rules for compilation and clean up
- Write up a report summarizing your implementation and detailing the results (as explained below)

Each of these items are explained in the sections below. You may be asked to provide a demo to the TA showing the results.

2.2 Directions

- The project must be emailed to TA, Duloor Rao (dulloor@gatech.edu), by *February 2nd 11 : 59PM*
- You are expected to use the GTThreads package provided to you. For any issues with the library, please contact the TA, Duloor (dulloor@gatech.edu).
- If you have any questions, you are encouraged to bring them to the TA office hours. For more urgent queries, you can email the TA, Duloor (dulloor@gatech.edu). Please include the email subject "Advanced OS : Project query".

- The TAs are working on providing a wiki for discussions. You will be notified when ready.

3 Credit scheduler for GTThreads library

3.1 GTThreads library

GTThreads is a user-level thread library. Some of the features of the library are as follows :

- Multi-Processor support : The user-level threads(uthreads) are run on all the processors available on the system.
- Local runqueues : Each cpu has its own runqueue. The user-level threads (uthreads) are assigned to one of these runqueues at the time of creation. Part of the work in the project might involve using some metrics before assigning these uthreads the processor and/or run-time runqueue balancing.
- O(1) priority scheduler and co-scheduler : The library implements these two scheduling algorithms. The code can be used for reference. Priority hash tables used in the library can be used, with some hacking, for the purpose of credit scheduler. In particular, look at the functions "sched_find_best_uthread_group" and "sched_find_best_uthread".
- The code runs on helsinki. It should run on other CoC machines too.

3.2 Credit Scheduler

The credit scheduler is a proportional fair share CPU scheduler built from the ground up to be work conserving on SMP hosts. Please take a look at the following resources :

- XenWiki for Credit Scheduler (<http://wiki.xensource.com/xenwiki/CreditScheduler>)
- The Xen Credit CPU Scheduler (http://www.xen.org/files/summit_3/sched.pdf)
- Comparison of the Three CPU Schedulers in Xen (http://xen.org/files/xensummit_4/3schedulers-xen-summit_Cherkosova.pdf)

4 Results and Report

4.1 Threading the Matrix Multiplication

A very rudimentary code for multiplying matrices is provided in the matrix directory. The code generates matrices (with each entry as 1) and then creates

a number of threads to multiply the matrices. The output of the multiplication corresponds to calculating $C = A*B$. Each thread calculates a fraction of the rows in C by calculating the same stripe of rows in A times the entire matrix B . Use this code to write a function that multiplies its own matrix. This is to mean that each uthread is working on its own individual matrix.

4.2 Test Cases

To get full credit for results, you must execute the following test cases:

- Run 512 utthreads.
- Credits are ranging in {25, 50, 75, 100}. Note : 128 utthreads have same credits.
- Matrix sizes are ranging in {64,128,256,512}. Note : 128 utthreads work on the same matrix size, but only 32 of them have same credits.
- Collect the time taken (to the accuracy of micro-seconds) for each of these tasks.

4.3 Reporting and summarizing results

To get full credit for the write up, you must include the following:

- Write a report summarizing the implementation of the project.
- Summarize results for all the test cases above.
- Plot graphs for your results.
- Mention clearly the implementation issues in your final submission.

Active monitoring of a system can be done in one of the following ways :

5 Submission

You should include the following in a tarball or similar:

- all of your source files;
- a Makefile for compiling your code;
- your write up as previously described.

6 References

- XenWiki for Credit Scheduler (<http://wiki.xensource.com/xenwiki/CreditScheduler>)
- The Xen Credit CPU Scheduler (http://www.xen.org/files/summit_3/sched.pdf)
- Comparison of the Three CPU Schedulers in Xen (http://xen.org/files/xensummit_4/3schedulers-xen-summit_Chernosova.pdf)
- GTThreads implementation (Link to be uploaded)