

B. Tech. Summer Internship Report

201401138 - Dhaval Prajapati

201401162 - Dhaval Panjwani

Subject : **Multi Core Simulation For Mobiles using Graphite Simulator**

Mentors : **Prof. P.S.Kalyan Sasidhar (DA-IICT)**

Prof. Reshmi Mitra (IIIT-Vadodara)

Summer Internship Co-ordinator : **Prof. Puneet Bhateja (DA-IICT)**

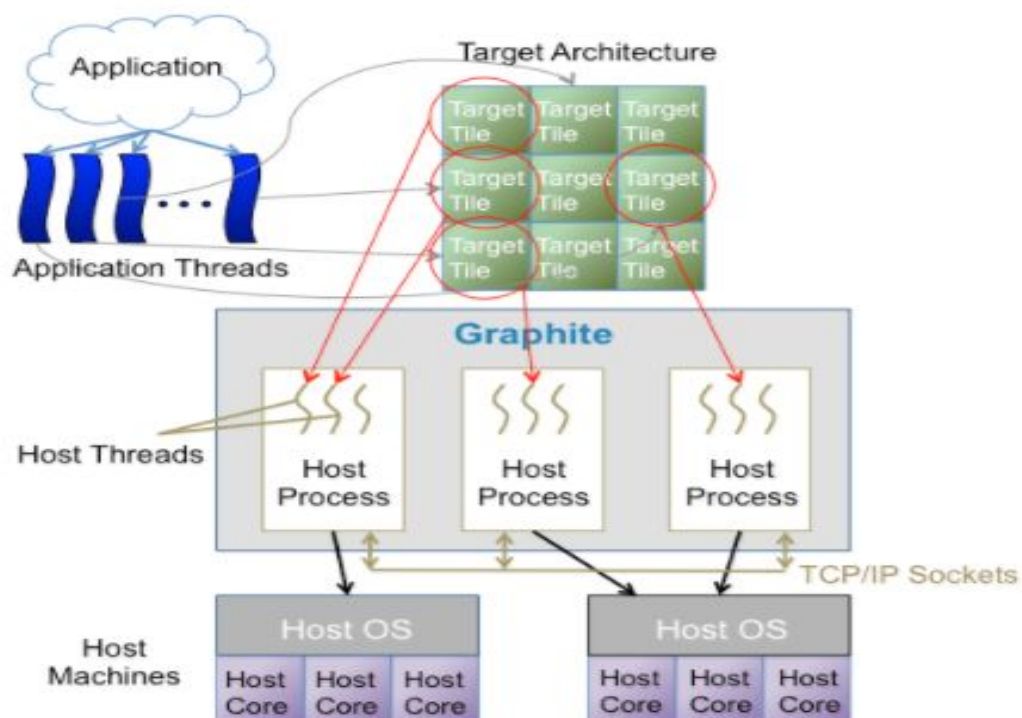
INDEX

Serial Number	Topic
1	Introduction
2	How to setup, install and use Graphite simulator in Ubuntu 12.04 !
3	Ping-Pong message sending between two processors
4	Bash Shell Script for automatically running Graphite over set of Configurations
5	System Information of mobile phones
6	Comparisons between Mobile Phones by simulating in Graphite

INTRODUCTION

At present times, at most of the places around the world the apps which are run on mobile phones are run on single processor. They are sequential given tasks to be individual processors out of all available processors on a mobile phone. Due to less awareness about parallel computing on mobile phones, most of the times the remaining processor stays idle for most of the time in the life cycle of a mobile phone. Our this exploration and research is based on the number of doors and opportunities parallel computing over mobile phones could give us for many situations . We have simulated the mobile phone multi-processor environment on a tool called called Graphite Simulator. This tool was contributed by Carbon Research group of Massachusetts Institute of Technology.

Graphite Multi Core Simulator :-



Graphite is an open-source, distributed parallel simulator for multicore architectures. Graphite is designed from the ground up for exploration of future multicore processors containing dozens, hundreds, or even thousands of cores. It provides high performance for fast design space exploration and software development.

Several techniques are used to achieve this including: direct execution, seamless multicore and multi-machine distribution, and lax synchronization. Graphite is capable of accelerating simulations by distributing them across multiple commodity Linux machines. When using multiple machines, it provides the illusion of a single process with a single, shared address space, allowing it to run off-the-shelf pthread applications with no source code modification.

Distributing a simulation :-

To distribute a Graphite simulation across multiple machines, edit the [general/num_processes] and [process_map] sections of the configuration file carbon_sim.cfg.

To run a Graphite simulation on two machines “server1.csail.mit.edu” and “server2.csail.mit.edu”, make the following changes to “carbon_sim.cfg”.

[general]

num_processes = 2

[process_map]

process0 = "server1.csail.mit.edu"

process1 = "server2.csail.mit.edu"

The above technique can also be used to run a Graphite simulation on two processes belonging to the same machine.

Setup, install and use Graphite simulator in Ubuntu 12.04

Graphite is an open-source, distributed parallel simulator for multicore architectures. Graphite is designed from the ground up for exploration of future multicore processors containing dozens, hundreds, or even thousands of cores. It provides high performance for fast design space exploration and software development. We tried basic things on graphite first for exploration. They are stepwise mentioned below:

To Install and Simulate Graphite tool you have to follow following step.

1. Install Graphite tool. Clone graphite locally from github from this [link](#). Preferable OS are ubuntu 12.0.4 and Debian Squeeze. [VMWare Player](#) is available for free for a variety of platforms and has an easy, streamlined process for installing these OSes. We only support 64-bit host and target architectures. (Note: Use only the versions of Ubuntu stated above. In using other newer versions of Ubuntu you will face the problem as Graphite won't build on newer versions.)
2. Download [Pin](#) from the pintool website Pin Downloads. Graphite currently works with version "62141".Untar Pin into a convenient directory. (Note: USE only this old version of Intel Pin stated above. Using other versions would not allow us to build the simulator)
3. Install the essential libraries needed for compiling (g++, make, libtool, etc...) and other required libraries through this commands.

\$ sudo apt-get update

\$ sudo apt-get install build-essential

\$ sudo apt-get install libtool automake autoconf autotools-dev

4. Graphite requires Boost version 1.48 on Ubuntu. For Squeeze, we recommend version 1.42 (substitute 42 for 48 in the command below). For this go through this command.

```
$ sudo apt-get install libboost1.48-dev libboost-filesystem1.48-dev  
libboost-system1.48-dev
```

5. Graphite requires BerkeleyDB to cache the power models obtained from CACTI and DSENT in a local disk database. This is to avoid recomputation for the same cache & network configuration across simulations. For this go through this command.

```
$ sudo apt-get install libdb-dev
```

6. If you are developing on Graphite, then you will probably want to use version control. Graphite uses git. To install this go through command .

```
$ sudo apt-get install git-core
```

7. Grab a tarball of the master branch from github, extract it and enter the directory.
8. Graphite uses git for source control. If you are unfamiliar with git, this site (github) provides a good introduction. The first step is to install git. Next, clone the repository:

```
$ git clone git://github.com/mit-carbon/Graphite.git [source directory]  
$ cd [source directory]
```

9. Building the simulator :- First, however we must configure a few details in the build system to configure paths of installed dependencies. Make the following

change to the file "Makefile.config": Set PIN_HOME to point to the directory where you untarred Pin.

10. Now simply run "make" in the graphite directory to build the simulator.

\$ make

To test your build, run the following command and watch for errors (warnings are OK):

\$ make ping_pong_app_test

If you get the code passes with no errors and returned normally, then the code has run successfully on the simulator. Warnings are okay.

You can also define the Numbers of Cores and Numbers of Processes(Machines) for Your Application.

Ex:

make ping_pong_app_test CORES=16 PROCS=2

This will run the ping_pong test with 16 cores and distribute the simulation across two machines.

Ping-Pong message sending between two processors

The ping pong app sends the message Ping to another active processor which is waiting and is ready to receive messages. Then after receiving, the processor 2 would reply back with Pong as an acknowledgment to the processor 1. Given below is the working of processors in graphite simulator for Ping-Pong app.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "carbon_user.h"

void* ping_pong(void *threadid);

int main(int argc, char* argv[]) // main begins
{
    CarbonStartSim(argc, argv);

    int num_threads = 2;
    carbon_thread_t threads[num_threads];

    for(unsigned int i = 0; i < num_threads; i++)
    {
        printf("Spawning thread: %d\n", i);
        threads[i] = CarbonSpawnThread(ping_pong, (void *) i);
    }

    for(unsigned int i = 0; i < num_threads; i++)
        CarbonJoinThread(threads[i]);

    printf("Finished running PingPong!.\n");

    CarbonStopSim();
    return 0;
} // main ends

void* ping_pong(void *threadid)
{
    int junk;
    int tid = (int)threadid;
    printf("Thread: %d spawned!\n", tid);

    CAPI_Initialize((int)threadid);

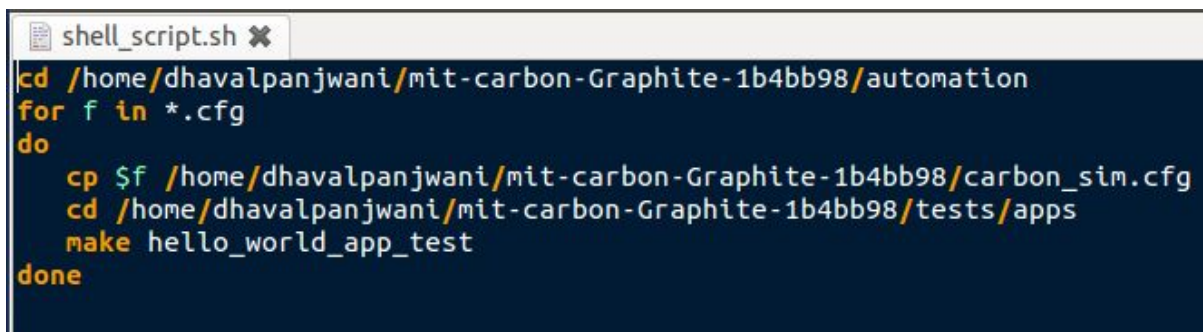
    //FIXME: there is a race condition on claiming a comm id...

    sleep(5);

    printf("sending.\n");
    CAPI_message_send_w((CAPI_endpoint_t) tid, !tid, (char*) &junk, sizeof(int));
    CAPI_message_receive_w((CAPI_endpoint_t) !tid, tid, (char*) &junk, sizeof(int));
}
```


Bash Shell Script for automatically running Graphite over set of Configurations

If we have a saved set of **carbon_sim.cfg** files in a folder to test them for an app, then we need to run Graphite for those configurations individually. But to save from that hectic monotonous work, we made an automatically running bash script which does the job for us. This script would take one configuration file at a time, will copy that into destination folder from where Graphite loads configurations. So every time a new configuration would be loaded into Graphite. Then Graphite would compile an app and run. And accordingly results will be stored in **sim.out** file. Below is the bash file for that purpose :



```
shell_script.sh ✕
cd /home/dhavalpanjwani/mit-carbon-Graphite-1b4bb98/automation
for f in *.cfg
do
    cp $f /home/dhavalpanjwani/mit-carbon-Graphite-1b4bb98/carbon_sim.cfg
    cd /home/dhavalpanjwani/mit-carbon-Graphite-1b4bb98/tests/apps
    make hello_world_app_test
done
```

System Information of mobile phones :-

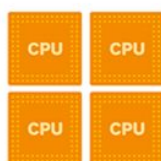
Following is the system specification for Iphone 7 plus mobile phone :-

System Information	
	iPhone9,4
Operating System	iOS 10.0.1
Model	iPhone9,4
Processor	ARM @ 2.23 GHz 2 processors
Processor ID	ARM
L1 Instruction Cache	64 KB
L1 Data Cache	64 KB
L2 Cache	3072 KB
L3 Cache	0 KB
Motherboard	D111AP
BIOS	
Memory	2998 MB

iPhone 7 Plus benchmark

Following is the system specification for chipset Kryo 280 which is used in Samsung S8 mobile phone :-

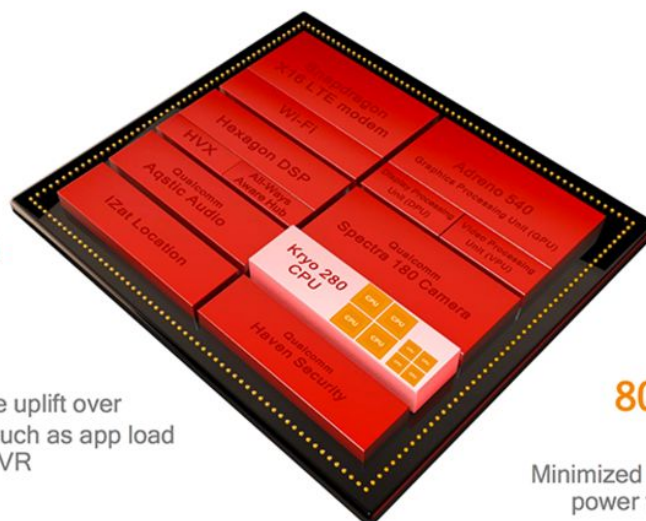
Kryo 280 Efficiency Cluster Optimization



Performance

Up to 2.45GHz
2MB L2

20% performance uplift over range of use cases such as app load time, web browsing, VR



Efficiency

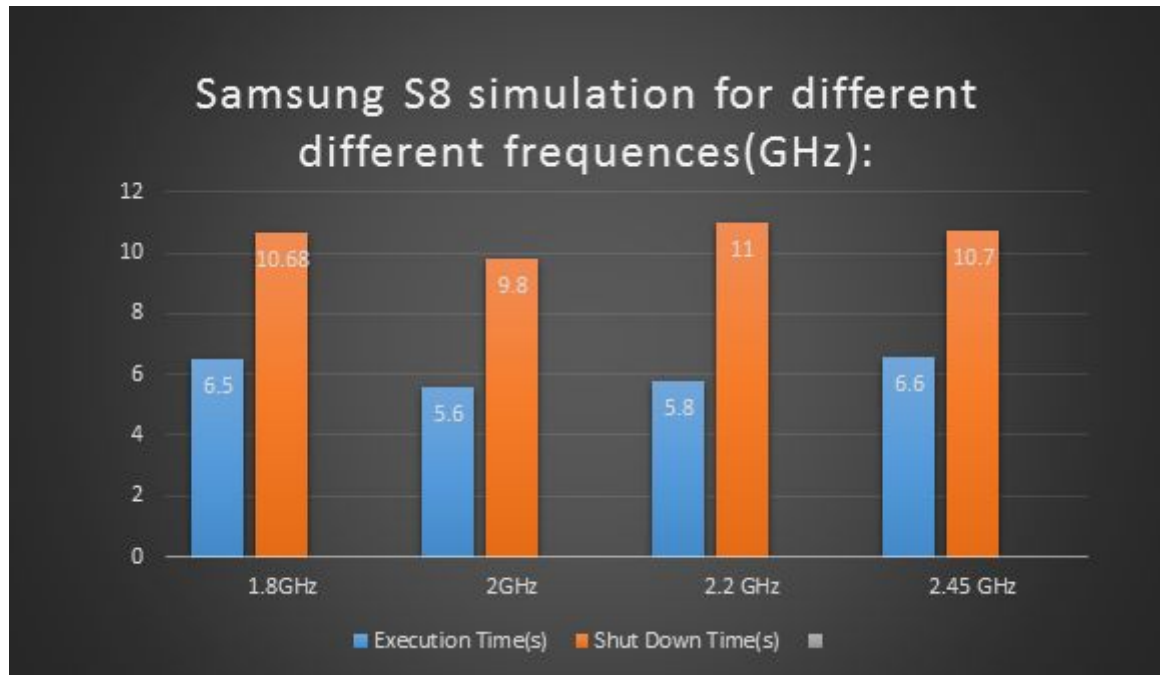
1.9GHz
1MB L2

80% of time is spent on efficiency cluster

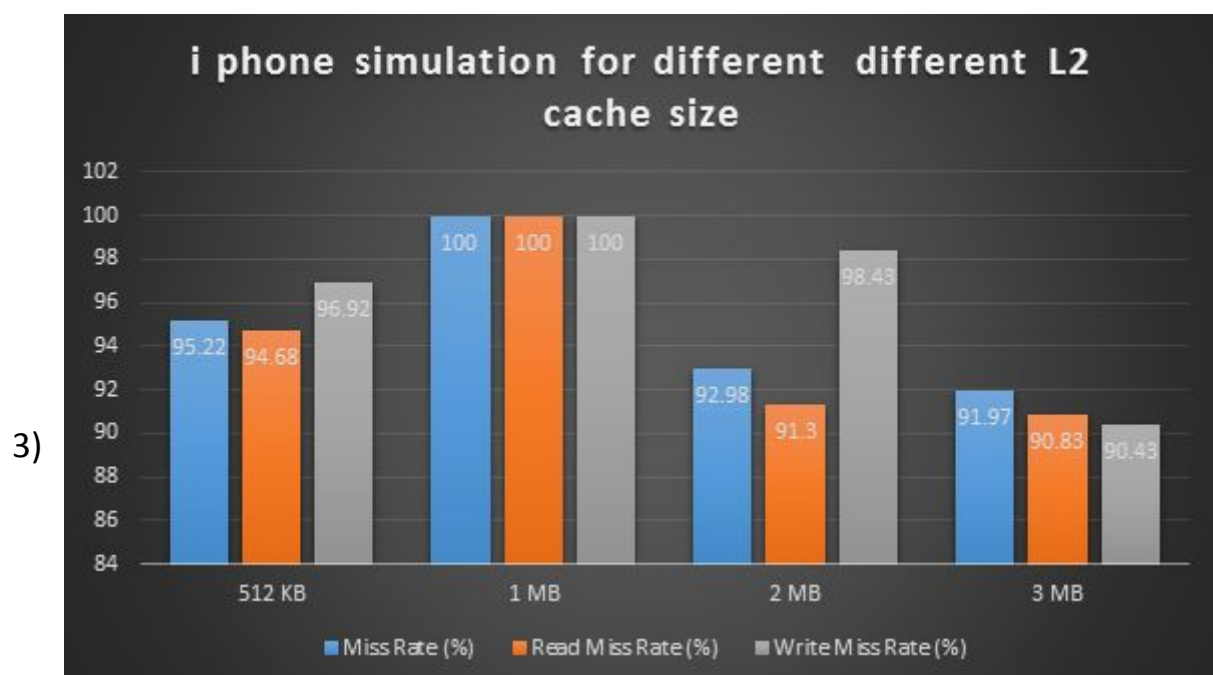
Minimized memory transaction power with larger L2 cache

Comparisons between Mobile Phones by simulating in Graphite

- 1) Processor Clock Speed Vs Execution time for Ping-Pong app (Samsung S8 benchmarks).

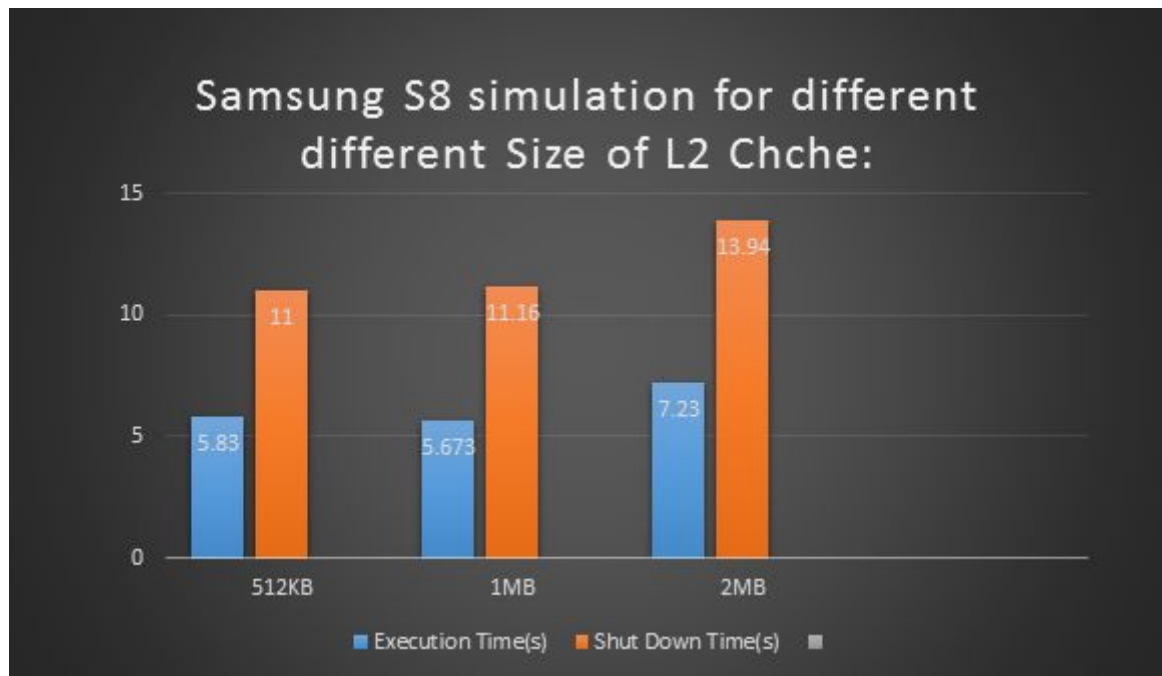


- 2) L2 Cache size vs Total Miss Rate , Read Miss Rate, Write Miss Rate (Iphone 7 Benchmarks).



Size of L2 cache Vs Execution time for Ping-Pong app (Samsung S8 benchmarks).

4)



Execution Time vs Shutdown Time for Different Different ialu and falu cycle costs for Ping-Pong app (Samsung S8 benchmarks).

