Datacenter Simulation Methodologies: MARSSx86 and DRAMSim2

Tamara Silbergleit Lehman, Qiuyun Wang, Seyed Majid Zahedi and Benjamin C. Lee

This work is supported by NSF grants CCF-1149252, CCF-1337215, and STARnet, a Semiconductor Research Corporation Program, sponsored by MARCO and DARPA.

Tutorial Schedule

Time	Topic
09:00 - 09:15	Introduction
09:15 - 10:30	**Setting up MARSSx86 and DRAMSim2**
10:30 - 11:00	Break
11:00 - 12:00	Spark simulation
12:00 - 13:00	Lunch
13:00 - 13:30	Spark continued
13:30 - 14:30	GraphLab simulation
14:30 - 15:00	Break
15:00 - 16:15	Web search simulation
16:15 - 17:00	Case studies

Agenda

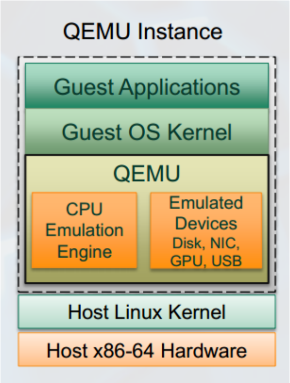
- Objectives
  - Understand simulator components
  - Be able to perform full system simulation
  - Be able to control simulation environment
- Outline
  - Create a disk image, qcow2 format
  - Configure and compile DRAMSim2, MARSSx86
  - Simulate programs
  - Create checkpoints
  - Simulate from checkpoints
  - Parse results

Simulator Requirements

- Software
  - Full system simulation: evaluate software stack behavior
  - Fast and easy to use: simulate long running applications
  - Multithread, multiprogram: support complex workloads
- Architecture
  - x86 support: most servers use x86 architecture
  - Multicore support: servers have many cores
- Future
  - Heterogeneous simulation (e.g., processors, memories)

Full System Simulation Overview

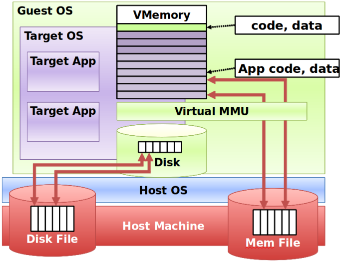
- Simulate complete software stack – applications, libraries, operating system.
- Use emulation engine to manage virtual environment



"MARSS: Micro Architectural System Simulator", ISCA Tutorial 2012 by Ghose *et al.*

A Little About Virtual Machines

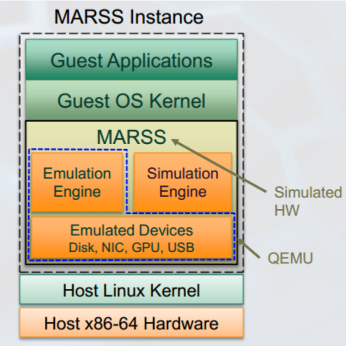
- Physical machine is host
- Emulated machine is guest
- Guest and host communicate in many ways
- Virtual machine disk is a file in the host
- Checkpointing saves disk state at any point in time



"A Survey of Virtual Machine Research" by Landon Cox (Duke CS510)

MARSSx86 Overview

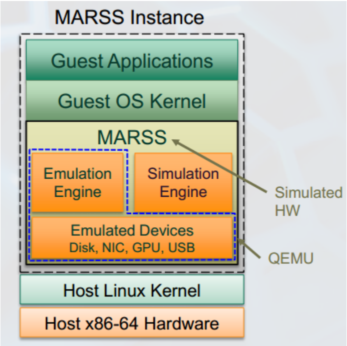
- PTLsim, QEMU Collaboration
- QEMU is emulator engine
- PTLsim is processor simulator
  - Detailed pipeline, cache simulation
  - Simple memory controller interface



"MARSS: Micro Architectural System Simulator", ISCA Tutorial 2012 by Ghose *et al.*

QEMU Overview

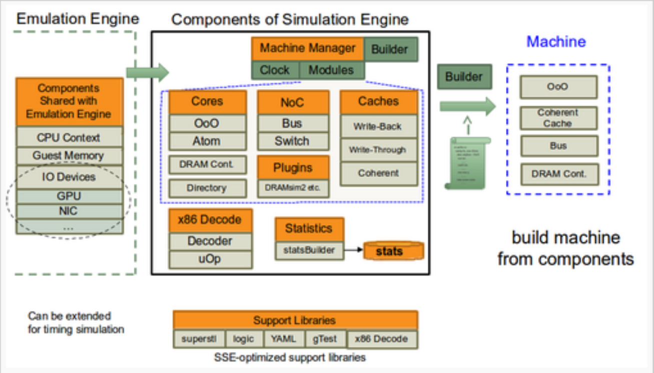
- Fast, easy to use emulator
- Uses dynamic binary translation
    - Translate instructions to C code
    - Compile C code for host
- QEMU emulates devices for functionality only. No performance estimates.



"MARSS: Micro Architectural System Simulator", ISCA Tutorial 2012 by Ghose *et al.*

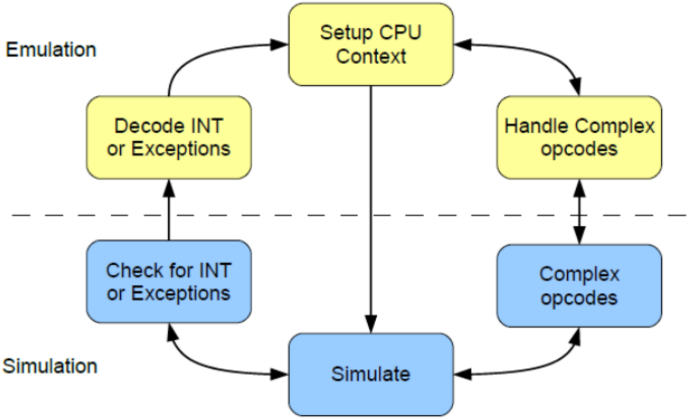
PTLsim Overview

- Cycle-accurate core, cache simulator
- Event-based simulation
- Specify microarchitecture in configuration files



"MARSS: Micro Architectural System Simulator", ISCA Tutorial 2012 by Ghose *et al.*

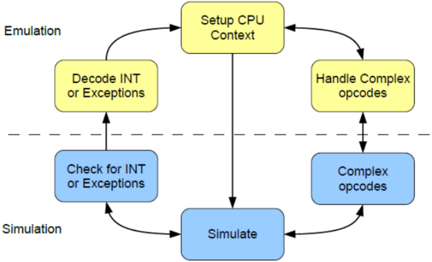
MARSSx86 Execution Flow



MARSSx86 (http://marss86.org)

MARSSx86 Execution Flow

- In simulation mode, PTLsim checks for interrupts, exceptions
- PTLsim saves its state, transfers control to QEMU
- In emulation mode, QEMU handles interrupt, returns control to PTLsim
- PTLsim restores state, continues execution

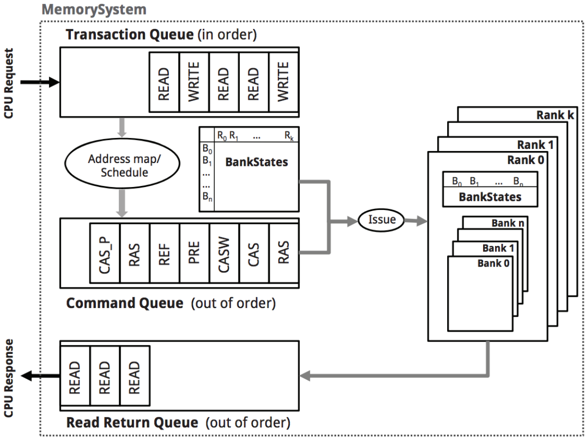


MARSSx86 (http://marss86.org)

DRAMSim2 Overview

- Simulates memory system in detail
- Simulates diverse memory technologies
- Specifies device details in configuration files
  - Scheduling policies
  - Addressing modes
  - Row buffer management policies

DRAMSim2 Overview



P. Rosenfeld *et al.* "DRAMSim2: A Cycle Accurate Memory System Simulator" CAL 2010

Putting All Together

- QEMU handles interrupt, exceptions, complex opcodes
- PTLsim simulates datapath, caches
- PTLsim sends memory requests to DRAMSim2

Questions?

Datacenter Simulation Methodologies
Getting Started with MARSSx86, DRAMSim2

Tamara Silbergleit Lehman, Qiuyun Wang, Seyed Majid Zahedi
and Benjamin C. Lee

This work is supported by NSF grants CCF-1149252, CCF-1337215, and STARnet, a Semiconductor Research Corporation Program, sponsored by MARCO and DARPA.

Agenda

- Objectives
  - Understand simulator components
  - Be able to perform full system simulation
  - Be able to control simulation environment
- Outline
  - Create a disk image, qcow2 format
  - Configure and compile DRAMSim2, MARSSx86
  - Simulate programs
  - Create checkpoints
  - Simulate from checkpoints
  - Parse results

Getting Started With MARSSx86

- Libraries needed:

```
git g++ scons zlib1g-dev libsdl1.2-dev
   libsdl1.2debian qemu
```

- Connect to the remote computer (ssh). You should have received login information.

- Get MARSSx86 source code:

```
$ git clone https://github.com/dramninjasUMD/
   marss.dramsim.git
```

- Get DRAMSim2 source code:

```
$ git clone git://github.com/dramninjasUMD/
   DRAMSim2.git
```

Creating a Disk Image

- The following instructions are just for illustration purposes. For today's tutorial we will use an already prepared image.
- Create a 10 GB qcow2 image:

```
$ qemu-img create -f qcow2 micro2014.qcow2 10
   G
```

- Install the operating system on the image:

```
$ qemu-system-x86_64 -m 4G -drive file=
   micro2014.qcow2,cache=unsafe -cdrom mini.
   iso -boot d -k en-us
```

  - On the installation menu choose the command-line install
    Note: This will take approximately 25 minutes.

Creating a Disk Image

- Once the operating system is installed re-run QEMU to prepare the virtual machine to run with PTLsim.

```
$ qemu-system-x86_64 -m 4G -drive file=
    micro2014.qcow2,cache=unsafe -k en-us -
    nographic
```

- Change the root password and login as root.

```
# sudo passwd root
# su
```

- Create file /etc/init/ttyS0.conf to be able to run simulations with a script:

```
# ttyS0 - getty
#
# This service maintains a getty on ttyS0 from the point the system is
# started until it is shut down again.

start on stopped rc RUNLEVEL=[012345]
stop on runlevel [!012345]

respawn
exec /sbin/getty -L 115200 ttyS0 vt102
```

- Open the tty port

```
# start ttyS0
```

Creating a Disk Image

- Open /etc/default/grub and modify it to look as below:

```
File Edit Options Buffers Tools Conf Help
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=1
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash rootdelay=200"
GRUB_CMDLINE_LINUX=""
```

- After closing the file update grub and power down the virtual machine:

```
# update-grub
# poweroff
```

DRAMSim2 Configuration

- Change into the DRAMSim2 directory
- DRAMSim2 uses system.ini to specify the system configuration parameters
- Open system.ini.example and save it as system.ini

```
; COPY THIS FILE AND MODIFY IT TO SUIT YOUR NEEDS
NUM_CHANS=1
JEDEC_DATA_BUS_BITS=64
TRANS_QUEUE_DEPTH=32
CMD_QUEUE_DEPTH=32
EPOCH_LENGTH=100000
ROW_BUFFER_POLICY=open_page
ADDRESS_MAPPING_SCHEME=scheme2
SCHEDULING_POLICY=rank_then_bank_round_robin
QUEUING_STRUCTURE=per_rank

;for true/false, please use all lowercase
DEBUG_TRANS_Q=false
DEBUG_CMD_Q=false
DEBUG_ADDR_MAP=false
DEBUG_BUS=false
DEBUG_BANKSTATE=false
```

DRAMSim2 Configuration

- The simulated device can be configured with an ini file.
- There are many ini files to choose from provided by the DRAMSim2 team in the ini directory.
- We will use ini/DDR3_micron_8M_8B_x16_sg15.ini

```
NUM_BANKS=8
NUM_ROWS=16384
NUM_COLS=1024
DEVICE_WIDTH=16

;in nanoseconds
;#define REFRESH_PERIOD 7800
REFRESH_PERIOD=7800
tCK=1.5 ;*
```

- Build the shared library to be used by MARSSx86:

```
$ make libdramsim.so
```

side note: for debugging add DEBUG=1 to the command.

MARSSx86 Configuration

- Change into marss.dramsim directory and open the machine configuration file: config/default.conf
- In this file we can import configuration files

```
# Import files that define various core/caches
import:
  - ooo_core.conf
  - atom_core.conf
  - l1_cache.conf
  - l2_cache.conf
  - moesi.conf
```

- We can specify many machine configurations
- To select which one to simulate, use the machine option in the simulation configuration file.

MARSSx86 Custom Configuration

- Below is the single core configuration example.

```yaml
machine:
  # Use run-time option '-machine [MACHINE_NAME]' to select
  single_core:
    description: Single Core configuration
    min_contexts: 1
    max_contexts: 1
    cores: # The order in which core is defined is used to assign
           # the cores in a machine
      - type: ooo
        name_prefix: ooo_
        option:
            threads: 1
```

- We will create a new configuration file to add microarchitectural details about the core and caches.
- We have provided an example configuration file. Open ~/custom.conf

MARSSx86 Custom Configuration

- Details about configuration file
  - Core section

```
core:
  ooo_custom:
    base: ooo
    params:
      ISSUE_WIDTH: 8
      MAX_PHYS_REG_FILE_SIZE: 196
      PHYS_REG_FILE_SIZE: 196
```

  - Cache section

```
cache:
  l1_32K_moesi_custom:
    base: moesi_cache #or mesi_cache
    params:
      SIZE: 32K
      LINE_SIZE: 64 # bytes
```

  - Memory section

```
memory:
  custom_global_dir_cont:
    base: global_dir
  custom_dram_cont:
    base: simple_dram_cont
```

MARSSx86 Custom Configuration

- More details about the configuration file
  - Machine section specifies number threads per core, which core, cache and memory controller to use

```
machine:
  custom:
    description: Custom Configuration
    min_contexts: 1
    cores:
      - type: ooo_custom
```

  - Within the machine section we can also specify the connections between all the components

```
    interconnects:
      - type: p2p
        connections:
          - core_$: I
            L1_I_$: UPPER
          - core_$: D
```

- More information available on the MARSSx86 web site:
  http://marss86.org/~marss86/index.php/Machine_Configuration

Simulation Configuration

- Simulation configuration parameters are specified through a file (micro2014.simcfg).

```
-logfile micro2014.log
#-run
-machine custom
-corefreq 4G
-stats micro2014.yml
#-kill-after-run -quiet
-dramsim-device-ini-file ini/
   DDR3_micron_8M_8B_x16_sg15.ini
-dramsim-system-ini-file system.ini
-dramsim-results-dir-name micro2014_dramsim
```

Compiling MARSSx86

- Build MARSSx86 with the custom configuration file and 4 cores:

```
$ scons -Q c=4 config=/hometemp/userXX/custom
    .conf dramsim=/hometemp/userXX/DRAMSim2
```

`Note: for debugging add` *`debug=2`*

- Previous command produces a new QEMU binary that integrates PTLsim into it.

Running MARSSx86

- Run MARSSx86 with the simulation configuration file:

```
$ ./qemu/qemu-system-x86_64 -m 4G -drive file
    =/hometemp/userXX/micro2014.qcow2,cache=
    unsafe -nographic -simconfig /hometemp/
    userXX/micro2014.simcfg
```

PtlCalls

- PtlCalls is the interface between PTLsim and QEMU.
- Many different functions:
  - ptlcall_switch_to_sim(): Goes into simulation mode.
  - ptlcall_checkpoint_and_shutdown(chkpt name): Takes a snapshot of the vm and shuts down.
  - ptlcall_switch_to_native(): Goes into emulation mode.
  - ptlcall_kill(): Terminate the simulation.
- Copy the file ptlcalls.h from the ptlsim/tools directory

```
# scp username@hostname://hometemp/userXX/
   marss.dramsim/ptlsim/tools/ptlcalls.h .
```

- Create 3 binaries for start_sim, stop_sim and kill_sim

Duke UNIVERSITY

```c
//start_sim.c
#include <stdlib.h>
#include <stdio.h>
#include "ptlcalls.h"

int main(int argc, char ** argv){
  printf("Starting simulation\n");
  ptlcall_switch_to_sim();
  return EXIT_SUCCESS;
}
```

```c
//stop_sim.c
#include <stdlib.h>
#include <stdio.h>
#include <ptlcalls.h>

int main(int argc, char ** argv){
  printf("Stopping simulation\n");
  ptlcall_switch_to_native();
  return EXIT_SUCCESS;
}
```

```c
//kill_sim.c
#include <stdlib.h>
#include <stdio.h>
#include <ptlcalls.h>

int main(int argc, char ** argv){
  printf("Shutting down simulation and vm\n");
  ptlcall_kill();
  return EXIT_SUCCESS;
}
```

Makefile

```
#Makefile
all: start_sim stop_sim kill_sim helloWorld
start_sim: start_sim.c ptlcalls.h
gcc -std=gnu99 -D_GNU_SOURCE -O3 -o $@ start_sim
    .c
stop_sim: stop_sim.c ptlcalls.h
gcc -std=gnu99 -D_GNU_SOURCE -O3 -o $@ stop_sim.
    c
kill_sim: kill_sim.c ptlcalls.h
gcc -std=gnu99 -D_GNU_SOURCE -O3 -o $@ kill_sim.
    c
helloWorld: helloWorld
gcc -std=gnu99 -D_GNU_SOURCE -O3 -o $@
    helloWorld.c
clean:
-rm -f start_sim stop_sim kill_sim helloWorld *~
```

Running MARSSx86

- Create a simple program (helloWorld.c)

```
//helloWorld.c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char ** argv){
  printf("Hello World\n");
  return EXIT_SUCCESS;
}
```

- Compile and run the program with start_sim, stop_sim and kill_sim.

```
# ./start_sim; ./helloWorld; ./stop_sim
#
# ./start_sim; ./helloWorld; ./kill_sim
```

About Checkpoints

- Checkpoints are snapshots of the qcow2 image.
- Saves the state of your machine at a particular point in time.
- To load a virtual machine from a checkpoint add "-loadvm checkpoint_name" to the MARSSx86 command
- Checkpoints are hardware configuration dependent (number cores, cache sizes, etc)

How to Checkpoint

- There are 3 ways of creating checkpoints:
  - Create checkpoint from the command line within QEMU (we will see this during the WebSearch presentation)
  - Embed ptlcall function calls within the source code
  - Use a script that uses either the first or second method to create multiple checkpoints (batch mode)
- There are 2 ways of running from checkpoints:
  - Add *-loadvm checkpointname* option to the QEMU command
  - Use a script to run multiple simulations (batch mode)

Creating Checkpoints: PtlCall Function Call

- Add a PtlCall to create a checkpoint inside the source code:

```
//helloWorld.c
#include <stdlib.h>
#include <stdio.h>
#include "ptlcalls.h"

int main(int argc, char ** argv){
  char * chk_name=getenv("CHECKPOINT_NAME");
  if(chk_name != NULL){
    printf("Creating checkpoint with name %s\
        n",chk_name);
    ptlcall_checkpoint_and_shutdown(chk_name)
        ;
  }
  printf("Hello World\n");
  ptlcall_kill();
  return EXIT_SUCCESS;
}
```

Creating Checkpoints: PtlCall Function Call

- Run the program again after setting the environment variable

```
# export CHECKPOINT_NAME=helloWorld
# ./helloWorld
```

- Now the checkpoint was created within the source code.

```
PTLCALL type PTLCALL_CHECKPOINT
MARSSx86::Creating checkpoint helloWorld
MARSSx86::Checkpoint helloWorld created
MARSSx86::Shutdown requested
```

Checkpoint Management

- Check the checkpoint was created:

```
$ qemu-img info /hometemp/userXX/micro2014.
    qcow2
```

```
image: ../micro2014.qcow2
file format: qcow2
virtual size: 10G (10737418240 bytes)
disk size: 2.2G
cluster_size: 65536
Snapshot list:
ID        TAG                 VM SIZE                DATE       VM CLOCK
1         helloWorld             324M 2014-10-21 13:18:56   01:00:49.718
```

- Delete checkpoint:

```
$ qemu-img snapshot -d helloWorld /hometemp/
    userXX/micro2014.qcow2
```

Creating Checkpoints: Batch

- A Python script to create checkpoints is provided with the MARSSx86 distribution code
- We provided a simplified one: /checkpoint_script.py
- Modify the user variable to match your username
- We added the commands needed for helloWorld checkpoint as shown below.

```
#HelloWorld
bench='helloWorld'
pre_command = "make clean; make ; export
   CHECKPOINT_NAME=\"%s\"\n" % (bench)
cmd = "./helloWorld"
bench_dict = {'name' : bench, 'command' : '%s
   \n%s\n' % (pre_command, cmd) }
check_list.append(bench_dict)
```

Creating Checkpoints: Batch

- Copy the provided ~/checkpoint_script.py into the marss.dramsim/util/ directory

```
$ cp ../checkpoint_script.py util/.
```

- Run script:

```
$ ./util/checkpoint_script.py
```

Simulating from Checkpoints: Direct

- Make sure the simulation configuration file has the run and kill commands: /hometemp/userXX/micro2014.simcfg

```
-logfile micro2014.log
 -run
...
 -kill-after-run -quiet
...
```

- Launch the simulation from the checkpoint

```
$ ./qemu/qemu-system-x86_64 -m 4G -drive file
    =/hometemp/userXX/micro2014.qcow2,cache=
    unsafe -nographic -simconfig /hometemp/
    userXX/micro2014.simcfg -loadvm helloWorld
     -snapshot
```

Simulating from Checkpoints: Batch

- The Python script to run from checkpoints needs a cfg file to specify the simulation parameters
- Open ~/util.cfg
- Update the user name

```
[DEFAULT]
user='userXX '
marss_dir = /hometemp/%(user)/marss.dramsim
```

- Copy ~/util.cfg file into the util/ directory inside the marss.dramsim directory

```
$ cp ../util.cfg util/.
$ emacs -nw util/util.cfg
```

Simulating from Checkpoints: Batch

- util/run_bench.py has been provided with the MARSSx86 distribution
- Command to run the script:

```
$ ./util/run_bench.py micro2014 -d
    micro2014_stats -c util/util.cfg --chk-
    name=helloWorld
```

MARSSx86 Results

- Open micro2014_stats/test.yml

File Edit Options Buffers Tools Help


```
---
base_machine:
  ooo_0_0:
    cycles: 74056
    iq_reads: 61817
    iq_writes: 39517
    iq_fp_reads: 0
    iq_fp_writes: 0
    dispatch:
      width: [62604, 983, 1163, 1016, 8290]
      opclass:
        logic: 8195
        addsub: 10722
        addsubc: 0
        addshift: 496
```


- Script to parse yml files:

```
$ ./util/mstats.py -y --flatten -n
   base_machine::ooo_custom_0_0.*::cycles -t
   total  micro2014_stats/helloWorld.yml
```

DRAMSim2 Results

- Open
  ../DRAMSim2/results/dramsim_helloWorld/DDR3_micron_8M_8B
  _x16_sg15/4GB.1Ch.8R.scheme2.open_page.32TQ.32CQ.RtB.pRank.vis

```
File Edit Options Buffers Tools Help
!!SYSTEM_INI
NUM_CHANS=1
JEDEC_DATA_BUS_BITS=64
TRANS_QUEUE_DEPTH=32
CMD_QUEUE_DEPTH=32
EPOCH_LENGTH=100000
USE_LOW_POWER=true
TOTAL_ROW_ACCESSES=4
ROW_BUFFER_POLICY=open_page
SCHEDULING_POLICY=rank_then_bank_round_robin
ADDRESS_MAPPING_SCHEME=scheme2
QUEUING_STRUCTURE=per_rank
DEBUG_TRANS_Q=false
DEBUG_CMD_Q=false
DEBUG_ADDR_MAP=false
DEBUG_BANKSTATE=false
DEBUG_BUS=false
DEBUG_BANKS=false
DEBUG_POWER=false
VIS_FILE_OUTPUT=true
VERIFICATION_OUTPUT=false
NUM_RANKS=8
!!DEVICE_INI
NUM_BANKS=8
NUM_ROWS=8192
NUM_COLS=1024
DEVICE_WIDTH=16
REFRESH_PERIOD=7800
```

Agenda

- Objectives
  - Understand simulator components
  - Be able to perform full system simulation
  - Be able to control simulation environment
- Outline
  - Create a disk image, qcow2 format
  - Configure and compile DRAMSim2, MARSSx86
  - Simulate programs
  - Create checkpoints
  - Simulate from checkpoints
  - Parse results

For more information on MARSSx86 visit
http://marss86.org/~marss86/index.php/Home

For more information on DRAMSim2 visit
http://www.eng.umd.edu/~blj/dramsim/

Tutorial Schedule

Time	**Topic**
09:00 - 09:15	Introduction
09:15 - 10:30	Setting up MARSSx86 and DRAMSim2
10:30 - 11:00	**Break**
11:00 - 12:00	Spark simulation
12:00 - 13:00	Lunch
13:00 - 13:30	Spark continued
13:30 - 14:30	GraphLab simulation
14:30 - 15:00	Break
15:00 - 16:15	Web search simulation
16:15 - 17:00	Case studies

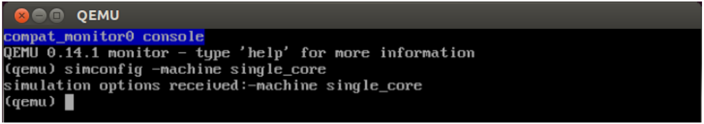
Backup Slides

- The following command requires display redirection (ssh -X option)
    - If using Ubuntu do not need anything additional
    - If using MacOS need to install xQuartz
- Run MARSSx86 with the following command (with graphics, display redirection required):

```
$ ./qemu/qemu-system-x86_64 -m 4G -drive file
    =/hometemp/userXX/micro2014.qcow2,cache=
    unsafe -simconfig /hometemp/userXX/
    micro2014.simcfg
```

Running MARSSx86

- QEMU has a control console that you can switch to with: Ctrl+Alt+2 (only with graphics mode)
- In the control console you can modify the simulation environment.
    - For example you can switch the machine being simulated:



- Press ctrl+alt+1 to go back to the virtual machine console.