Minerva User Manual

by

Rohith R

January 13, 2017

Contents

1. Introduction

All information about Minerva is given in the paper and the website.

2. Installation

Minerva has been tested to work on linux based systems. Latest test was done in a freshly installed Ubuntu 14.04.03 LTS 64 bit.

2.1 Install prerequisite software

Minerva is an extension to Snipersim and uses it as the base x86 simulator. Snipersim requires the following packages to be installed :

- zlib1g-dev
- libbz2-dev
- g++ (choose version 4.X, http://askubuntu.com/questions/26498)
- libsqlite3-dev
- libboost-dev
- m4
- xsltproc
- libx11-dev
- libxext-dev
- libxt-dev
- libxmu-dev
- libxi-dev
- gfortran

All the above packages can be installed using the command :

```
$ sudo apt-get install <package-name> # Use install_dependencies.sh
```

2.2 Download and Install Minerva

Minerva can be downloaded from http://wwww.example.com. Extract the zip file to a folder called `minerva` and place it in the appropriate folder you want to use Minerva from [1]. Now we can proceed to install Minerva. Open a terminal and change the directories to the folder where Minerva was extracted. All the commands to install have to be executed from the root folder `minerva`.

```
$ chmod +x -R * # Setting everything as executable
$ ./install_pin.sh
$ make -j 4
# Here 4 is the number of cores on the system. (System dependent)
```

[1]Minerva already comes with the pin tool kit PIN 71313 downloaded from https://software.intel.com/en-us/articles/pin-a-binary-instrumentation-tool-downloads

The installation can be verified by running a test run.

```
$ cd test/fft
$ make run
```

2.3 Install benchmarks

Minerva currently supports both the `splash2` and `parsec` benchmarks. Benchmarks are already downloaded and placed in the `benchmarks` folder inside Minerva's root folder. to install the benchmarks use the following commands :

```
$ python set_environment.py
$ source ~/.bashrc
$ ./install_benchmarks.sh
```

To verify the installation of the benchmarks, run the following command :

```
$ cd benchmarks
$ chmod +x verify_benchmark_installation.sh
$ ./verify_benchmark_installation.sh
```

We have completed the installation of Minerva and are now ready to use it for testing the reliability of the many core system. Before starting the simulation, we have to configure the run by specifying the details of the applications to be run, number of cores etc. This can be done by editing a file `config.py`. Details of how Minerva can be configured in different ways to test out different optimizations is explained in detail in the next sections.

3. Default Options

Before configuring Minerva and starting the simulations, it is important to understand the default options available with the tool.

The 2 simulation options available are :

- Single run of the specified benchmarks
- Periodic Workload Generation

The default scheduling options are :

- Round-Robin
- Temperature minimization using DVFS (To be used with Single Run option)
- A Hierarchical Approach for Enhancing Lifetime Reliability of Many-core Systems

The default reliability mechanisms available are :

- Electromigration
- NBTI
- TDDB
- Thermal Cycling

These options have to be conveyed to Minerva through the config file provided. The details of the config file and how to change it are explained in the next chapter.

4. Configuring the run

Before each running a set of applications, Minerva has to be configured and the configuration options are conveyed to Minerva by `config.py` file in the Minerva's root folder. The default configuraion file provided is explained below in a step-by-step manner and can be configured according to the user's needs.

Number of Cores

```
# Number of cores of the many core-sytem that is to be simulated.
# Should be a power of 2
num_cores = 8
```

Grid Dimensions

```
# Specify the Grid dimemsions. The processor is modelled as a grid X cross Y grid
# Example for a 16 core processor, X=4,Y=4 is one possible configuration
# Note X*Y should be equal to num_cores
X = 4
Y = 2
```

Operating Frequency

```
# Operating Frequency in mega hertz
frequency = 1000.0
```

List of applications

```
# List of Applications that you want to simulate
# Each application has a certain number of therads that it will spawn
# Specified as "benchmark_name-application_name-number_of_threads"
# Eg. app_list = ["splash2-fft-4","splash2-barnes-4"]
# Note : When a new benchmark is to be added, append to the list.
# The app_id of each app will be the same as specified here
app_list = []
app_list.append ("splash2-fft-4")
app_list.append ("splash2-fft-4")
```

Input Size

```
# Input Size is the problem size which the size of the data set used for input
# Possible values are "test","small","medium","large"
input_size = "test"
```

Reliability optimisation algorithm : Algorithm file is for advanced users who want to test out their own reliability algorithm.

If left blank then the default algorithm which optimises the Reliability is used. The path for this algorithm is `algorithm/RTApproach`. See the paper for the reference for this algorithm : .

For temperature minimisation put `"temp_min"` in this field. This option is for actual runtime intervention, and is effective when `runtime_intervention = True`. For a simple round-robin mapping specify `"round-robin"`

```
path_to_algorithm_file = "temp_min"
```

Runtime Intervention : Set this option as `"True"` if intervention is required while the applications are running. By default the option is False. If this option is `"True"`, then specify the time period after which the callback is to be provided. Time is in femtoseconds. **NOTE :** If `runtime_intervention` is `"False"`, specifying the `time_of_intervention` will have no effect.

```
runtime_intervention = True
time_of_intervention = 11234356789
```

Periodic Workloads : This option is to simulate the set of applications repeatedly. Note : If both the `runtime_intervention` and `periodic_benchmarks` are `"False"` then we return to original Snipersim without any extra interface.

```
# By default the option is "True"
# Number of epochs is the number of periods the simulation has to run
periodic_benchmarks = True
number_of_epochs = 1
```

Reliability Mechanisms

```
# Failure Mechanism is the one of many available options
# though which the Reliability will be evaluated
# Currently available option(s) are : "electromigration","nbti","tddb","tc"
failure_mechanism = "electromigration"
```

Epoch Length

```
# Epoch length is the time between the sucessive periods.
# The default value is 1*30*24*60*60 which is 1 month (30 days)
# Value is in seconds
epoch_length = float(1*30*24*60*60)
```

Visualizations

```
# Visualisations
# See the different types of graphs generated for each epoch
# Each option has the value 'True' or 'False' depending on whether the
# corresponding Visualisation is required

# 1. Temperature Heat Map
temperature_heat_map = True

# 2. MTTF Heat Map
mttf_heat_map = True

# 3. Min MTTF over time
# Compares the optimised approach and the normal scenario where there is no intervention
# This option allows the user to understand how optimal is the approach used
min_mttf_over_time = True

# 4. Average Power Line Graph
# This option is used to output the average power of each core after every epoch
avg_power_graph = True
```

This concludes the options that are available for configuration in Minerva. After chaning the options according to the needs of the researchers, user has to execute the python script `install_config.py` in the terminal.

```
$ python install_config.py
$ cd benchmarks
$ chmod +x new_run_b.sh
$ ./new_run_b.sh
```

This wills start the simulation of the benchmarks indicated for the required number of epochs.

5. Temperature, power & MTTF values

Minerva outputs the power and temperature after every specified interval of time in the config file. The power is logged by default after every 100 ns and the temperature is calculated using HotSpot after the specified interval of time. Since MTTF and parameters related to the reliability don't change as frequently as power and temperature, they are logged after the completion of every epoch.

5.1 Power File

The format of the power file for an 4 core system is :

```
C0      C1      C2      C3
24.3139 25.0371 24.989 25.1428
24.3139 25.0371 24.989 25.1428
24.3139 25.0371 24.989 25.1428
```

It contains many entries, **each entry is a space separated value of the powers of each core in Watts.**

The file will be named `temp-power-trace.ptrace` and placed in the `extra-files` folder. **Example files (inside `example-files/` folder) are provided with the installation for the users to understand the interfaces easily.**

5.2 Temperature File

The format of the temperature file for an 8 core system is :

```
C0 353.00
C1 353.41
C2 355.54
C3 355.55
C4 355.40
C5 355.67
C6 353.94
C7 354.00
```

Each line of the file contains an entry, which denotes the core number and the corresponding temperature value in Kelvin separated by space.

The file will be named `temp-steady-file.ttrace` and placed in the `extra-files` folder. **Example files (inside `example-files/` folder) are provided with the installation for the users to understand the interfaces easily.**

5.3 MTTF File

The format of the MTTF file for an 8 core system is :

```
6.7 4.5 5.6 7.1 3.2 4.8 6.3 5.9
```

The file contains the value of MTTF of each core in years separated by space.

The file will be named `mttf-file.txt` and placed in the `extra-files` folder. **Example files (inside `example-files/` folder) are provided with the installation for the users to understand the interfaces easily.**

All the values of MTTF, power and temperatures are logged after the end of the simulation in the `extra-files` folder under their respective folders.

5.4 Instruction Count File

To get the impact of optimization of DVFS, scheduling on the number of instructions executed, the instruction count of each core is logged along with the start and the end times of the interval. The format of the instruction count file is :

```
Number_of_instructions_executed Interval_Start_Time Interval_End_Time
```

The file will be named `ic-file.txt` and placed in the `extra-files` folder. **Example files (inside `example-files/` folder) are provided with the installation for the users to understand the interfaces easily.**

6. Interacting with scheduler and DVFS

Users can write programs to interact with the scheduler inside Minerva. This option can be used to override all the default scheduling algorithms provided with Minerva like round-robin, temperature minimization and MTTF optimization. User programs will be called by Minerva after every specified time period and at the end of every epoch. The user program can read the temperature, power and MTTF values of each of the cores from their respective files. All the files holding the statistics will be updated before the user program is invoked from Minerva.

6.1 Interacting with the scheduler

6.1.1 Design-time and Runtime Mapping

] Users can provide a mapping between application threads and the respective cores on which they are supposed to be pinned to. The format of the file is :

Table 6.1: An example of design time mapping of 2 apps on 8 cores

Core Id	App Id	Thread Id
0	0	0
1	-1	-1
2	1	0
3	1	1
4	-1	-1
5	-1	-1
6	0	1
7	-1	-1

Each entry of the mapping file contains three values separated by space, core id, application id, thread id. **An entry of `3 2 4` means that on the 3rd core [1], 2nd application's 4th thread is supposed to run.** A value of `-1` in the application id's field means that no thread is going to run on that particular core. For example an entry of `4 -1 -1` means that on the 4th core no application threads are scheduled.

A user program, when called by Minerva, is supposed to make the mapping file in this format after doing the optimization intended. The file to which the user program has to write is `runtime-mapping.txt`. An initial mapping of application threads to cores can be provided in the same format in the file `design-time-mapping.txt`. Both of these file are have to be located in the folder `extra-files`. **Example file are provided with the installation for the users to understand the interfaces easily.**

[1] All values are 0 based indexing

6.1.2 Re-scheduling in between the runs

Minerva also provides an option to re-schedule the threads during the runtime of the simulation to keep a check on temperature, power and other statistics. The threads from a source core can be moved to a destination core. The first line of the file should a either `0` or `1` depending on whether there is any change in the mappings or not. If `1` then, the subsequent file should contain entries of the form `source-core destination-core` separated by a space. The file should be named `mapping-file.txt` and placed in the `extra-files` folder. **Example files are provided with the installation for the users to understand the interfaces easily.**

6.2 Overriding the default scheduler

After building the desired scheduler with the optimization goal in mind, it can be integrated with Minerva by changing the `path_to_algorithm_file` field in the config file :

```
path_to_algorithm_file = ""
```

The same program is responsible for scheduling for both the type of schedulers in Section 6.1.1 and Section 6.1.2. **Minerva calls the user program from the C++ code using the `system()` function provided by linux. Users have to place their algorithm binary in the `algorithm` folder and provide the same command used to execute the algorithm from the terminal as the value of the field `path_to_algorithm_file`.**

For example, if the user has written a C++ program as a scheduler which produces a mapping, then he has to compile it, produce a binary and give the absolute path of the binary as the field value. If the name of the binary is `new_scheduler`, then the field will be :

```
path_to_algorithm_file = "MINERVA_ROOT_PATH/algorithm/new_scheduler"
```

If the scheduler is implemented in python, for example as `new_scheduler.py`, then the field should be :

```
path_to_algorithm_file = "python MINERVA_ROOT_PATH/algorithm/new_scheduler"
```

6.3 DVFS Control

Along with scheduler level control, Minerva also provides control over frequencies that the cores operate on at any instant of time. Users can specify the time interval after which they want a call back is given to the user-program which can change the frequencies of individual cores by using the interface provided by the tool. Before the control is passed on to DVFS program, the power values are logged and the temperature values are calculated which can be further used for some optimizations.

An example of DVFS file for an 8 core system is :

```
590 590 477 477 477 477 656 656
```

The values of the frequency are provided in MHz. The above values represent that core-0 will run with frequency 590 MHz, core-1 with 590 MHz, core-2 with 477 Mhz and so on.

A non-positive value corresponding to any core means that the core is a broken state. The file for DVFS is `dfvs-file.txt` and placed in the `extra-files` folder. **Example files (inside `example-files/` folder) are provided with the installation for the users to understand the interfaces easily.**

7. Process Variation File

Minerva as a reliability tool is also process variation aware. Process variation can affect the frequency, operating voltage etc. of each core differently and can play an important role in determining the MTTF of the many-core system. All the reliability models included in Minerva take the process variation of the cores into consideration while calculating the MTTF and ageing of the cores.

The process variation is communicated to Minerva by a file which contains space separated values of the process variation parameter (between 0 and 1) for each core. An example of a process variation file for a 8 core system is :

```
0.90    0.97    0.88    0.93    0.76    1.00    0.89    0.93
```

This indicates that the process variation parameter of the 0th core is 0.90, 1st core is 0.97 etc. These values are automatically passed on to Minerva and Snipersim through appropriate mechanisms.

The file should be named `pv-file.txt` and placed in the `extra-files` folder. **Example files (inside `example-files/` folder) are provided with the installation for the users to understand the interfaces easily.**

8. Implementing custom RMU

Expert user who want to override the default mechanisms which affect lifetime reliability can implement their own Reliability Management Unit (RMU). This can be done by first studying the existing RMU code provided and looking at how the `RMU_EM`, `RMU_NBTI` etc. have been implemented in Minerva. The code for these can be found in `common/RMU` folder.

8.1 Implementation

To implement a custom RMU, users have to override two function : `updateMTTF` and `updateAgeing`. After implementing the functions, the custom RMU is automatically available for use inside Minerva. Below is an example of the TDDB related mechanism is implemented :

File : `RMU_TDDB.h`

```
#ifndef __RMU_TDDB_H
#define __RMU_TDDB_H
#include "RMU.h"
/*
* These are constants used in the calculation of reliability using the TDDB mechanism
*/
class Constants_TDDB
{
        public:
                static constexpr double T_base = 345.0;
                static constexpr double TOTAL_TDDB_FITS = 800.0;

};
/*
* Class for calculating the TDDB related MTTF
* NOTE : We are not overriding the updateAging() because we are not simulating the aging.
*/
class CoreInfoTDDB : public CoreInfo
{
        double TDDB_fits;
        double TDDB_base_fits;
        double TDDB_inst;

        public:
                CoreInfoTDDB();
                ~CoreInfoTDDB();
                void setPVParam(double process_variation_parameter);
                void updateMTTF();
};
#endif
```

File `RMU_TDDB.cc` contains the implementation of the same :

```
#include "RMU_TDDB.h"

#include "simulator.h"
#include "config.hpp"

#include <cmath>

CoreInfoTDDB::CoreInfoTDDB()
 : CoreInfo()
 , TDDB_fits(0.0)
 , TDDB_base_fits(0.0)
 , TDDB_inst(0.0)

{

        // Initialize Constants
        // Actual Code can be found commom/RMU/RMU_TDDB.cc

}

CoreInfoTDDB::~CoreInfoTDDB()
{
}

void CoreInfoTDDB::setPVParam(double process_variation_parameter)
{
        pv_param = process_variation_parameter;
        frequency = pv_param * frequency;
        V = pv_param * V;
}

/*
* The reliability code is taken from RAMP 2.0.
*/

void CoreInfoTDDB::updateMTTF()
{

        // Put the actual code here.
        // Actual code can be found in commom/RMU/RMU_TDDB.cc

}
```

8.2 Plugin with Minerva

After the custom has been implemented by overriding the functions `updateMTTF` and `updateAgeing` the user has to let Minerva know aout this new RMU. To do this an entry has to be added to the `RMU.cc` file in the `common/RMU` folder. To add an RMU with the name `RMU_TDDB` as implemented above, add the following lines to the constuctor `RMU::RMU()` :

```
else if (failure_mehcanism == "tddb")
{
        for (int i=0;i<num_cores;i++)
                coreList.push_back(new CoreInfoTDDB());
}
```

This will allow the users to specify the option as `"tddb"` in the config file here :

```
# Failure Mechanism is the one of many available options
# though which the Reliability will be evaluated
# Currently available option(s) are : "electromigration","nbti","tddb","tc"
failure_mechanism = "electromigration"
```

This concludes the explanation of implementing a custom Reliability Management Unit into Minerva.

9. Graphs and animations

Minerva offers a variety of visualizations to indicate the state of the many-core system after every epoch. The values that are collected during the run of the workload are used to generate the graphs, heat-maps and the clips of various parameters. The plots also take into consideration the architectural floor-plan of the many-core system.

Some additional modules are required to be installed for the graphs to be plotted. They can be installed using the following commands :

```
$ sudo python -m pip install -U pip setuptools
$ sudo python -m pip install matplotlib
$ sudo apt-get install python-tk
```

To generate the graphs after running a simulation, execute the following command on the terminal :

```
$ python generate_graphs.py
```

All the graphs and animations are automatically saved in high-resolution formats in the `graphs/` directory.