Manifold 1.1 User Guide

He Xiao
hxiao@gatech.edu

April, 2016

Introduction

Manifold is a parallel discrete event simulation framework for simulation of modern multicore computer architectures. The software package consists of a parallel simulation kernel, a number of component models, and a few ready-to-use simulator programs that use the component models to build and simulate system models. Users can also port third-party components to Manifold and build system models with them. This user guide describes how to obtain Manifold source code, and how to build and run the simulator programs.

Overview

Manifold is designed for parallel simulation of multicore systems. The general simulation system is shown in Figure 1 below.

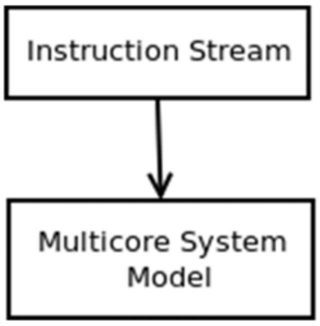

Figure 1. Manifold Simulation System

At run-time, instruction streams are fed to the multicore system model for simulation. Example source of instructions includes the QSim [1] multicore emulator. Components of the system model can be assigned to different host machines for parallel simulation.

The following are the general steps that a simulator program needs to follow to create a system model for simulation.

- Instantiating the various component models, such as processor, cache and so on.
- Connecting the components with Manifold links.
- Setting a simulation stop time.
- Starting the simulation by calling `Manifold::Run()`.

The simulator programs that are part of the distribution package can serve as examples for how to write simulator programs with Manifold. The component models that are included in the package can be used in building system models. The user can also port third-party components to Manifold and build system models using such components.

Major features of Manifold include the following:

- Supporting both sequential and parallel simulations.
- Supporting three simulation paradigms: discrete time event-driven simulation, time-stepped simulation, and mixed event-driven and time-stepped simulation.
- Supporting the QSim multicore emulator front-end.
- The Kitfox power, energy, thermal, and reliability modeling library for integration with multicore processor models.
- Standard interfaces between components allow mix-and-match of components.
- Open software architecture allows easy porting of third-party components.

Current Release

The current release is Manifold-1.1, which depends on the QSim-2.3.1 release. It supports both sequential and parallel simulation. The software is distributed as a source code package that contains the following:

- The parallel simulation kernel.
- Models:
  - SPX: a superscalar out-of-order processor model for x86_64 and aarch64.
  - Mcp-cache: a coherence cache model.
  - Simple-cache: a simple write-through cache model.
  - Iris: a cycle-level interconnection network model.
  - CaffDRAM: a DRAM controller model.
  - DRAMSim2: a port of the open-source DRAMSim2 model developed at University of Maryland.
  - *Simple-proc: a 1-IPC processor model (deprecated).*
  - *Zesto: a cycle-level x86 processor model (deprecated).*
- Simulator programs.
  - QsimProxy: uses QSim as proxy to generate instructions to drive manifold.

Testing and Portability

- Manifold has been tested on the PARSEC [2], SPLASH-2 [3] and graphBig [4] benchmarks.
- Manifold has been tested under the following operating systems:
  - Ubuntu 14.04.
  - Fedora release 22.
- Default installation script (setup.sh) works for Ubuntu 14.04.
- Manifold has been tested with Openmpi and MPICH2.

Source Code Directory Structure

The Manifold source code is organized as follows:

```
ROOT
  |... doc
  |... kernel
  |... models
  |     |... cache
  |     |     |... mcp-cache
  |     |     |... simple-cache
  |     |... cross
  |     |     |... mcp_cache-iris
  |     |... memory
        |     |     |... CaffDRAM
        |     |     |... DRAMSim2
  |     |... network
  |     |     |... iris
  |     |... processor
  |     |     |... simple-proc
  |     |     |... zesto
  |     |     |... spx
  |     |... qsim
  |     |     |... interrupt_handler
  |     |     |... proxy
  |... uarch
  |... util
  |... simulator
  |     |... smp
  |     |     |... common
  |     |     |... config
  |     |     |... QsimProxy
  |     |     |... QsimClient
  |     |     |... QsimLib
  |     |     |... Trace
```

where *ROOT* represents the root of the source tree. Cross directory contains the shared information between mcp-cache and iris network. Uarch defines the packet used for the iris network.

List of Models

Zesto: (deprecated, back compatible with QSim-0.1.5)

- A cycle-level x86 processor model. This model can accept instruction streams from three different sources: trace files, QSim server, and QSim library.
- Both in-order and out-of-order models are included.

SPX:

- Not as detailed as Zesto, SPX models all the important components of a modern superscalar processor. Current model supports only QSim proxy.

Simple-proc: (deprecated, back compatible with QSim-0.1.5)

- A simple 1-IPC (instruction per cycle) processor model. It can accept instruction streams from three different sources: trace files, QSim server, and QSim library.

Mcp-cache:

- Supports the MESI protocol.

Simple-cache:

- A simple write-through cache model. Currently it can only be used in a single-core model.

Iris:

- Supports ring and torus.
- Supports virtual channels.
- Supports two virtual networks.
- Supports flow-control between routers, between router and interface, and between interface and terminal.
- Supports single-flit packets.

CaffDRAM:

- Supports flow control between the memory controller model and the network model.

DRAMSim2:

- Detailed open-source dram model.

Simulator Directory

The simulator programs reside in smp directory. Programs allow one component to be replaced by another by simply changing a configure file in the smp/config directory. Current release only supports spx processor model, so users can use conf*_spx_*.cfg as a reference example.

The common code of the simulator programs is located in common, which includes the model initialization and connection. For example, proc_builder.{h,cc} initializes the processor cores using the parameters specified in config file, and connect the cores with L1 cache models via manifold kernel api (i.e. Connect function).

Build Process Overview

To build and run the simulator programs that are part of the software package, you will need to perform the following steps:

- [Optional] Install required packages.
- [Optional] Download and build QSim.
- Build Manifold libraries and simulator program(s).
- Configure and Run the simulators.

Current manifold version get instructions from QSim library through the manifold proxy components. Depending on which source you use, some of the steps above may be optional. The following explains each step in detail.

Install Required Packages

Before you proceed, you need to install the following required packages.

- **mpi:** We have tested with openmpi and mpich2, so these two packages are recommended.
- **libconfig++:** The simulators require this package. You should install the development version, such as libconfig++8-dev.
- **gcc & g++:** The multilib version of gcc should help avoid compilation errors. (apt-get install gcc-multilib)
- **zlib:** (apt-get install zlib1g-dev)

Building QSim

The current release of manifold uses QSim to get the instruction flows with QSim version 2.3 or higher, yet is not back compatible with earlier version of qsim (i.e. QSim-0.1.5). Detailed building instructions can be found in QSim user manual [5].

The QSim source code can be found from the following GitHub link from gtcasl repository.

```
https://github.com/gtcasl/qsim
```

Make sure the environment variable QSIM_PREFIX is set to the correct QSim directory before continue building manifold libraries and simulator. QSim installation directory should contain:

```
ROOT
  |... lib
      libqemu-qsim-a64.so
      libqemu-qsim-x86.so
      libcapstone.so
      libqsim.so
  |... include
      qsim.h
      qsim-*.h
      mgzd.h
      systemz.h
      xcore.h
      capstone.h
      platform.h
      <arch>.h
  |... bin
      qsim-fastforwarder
  |... state.n
  |... state.n.a64
```

where the state.n are the Linux state files for x86_64 architecture up to 64 cores, and the state.n.a64 are for arm 64 up to 8 cores. State files are needed by the Manifold simulator.

Building Manifold Libraries and simulator

Users can either download the latest stable version from the manifold website [6], or get the developing repo from github:

```
$ git clone https://github.com/gtcasl/manifold
```

The setup.sh script compiles and installs the manifold modules and simulator automatically for Ubuntu users. For users of other Linux distributions, make sure the required packages are installed in the first step. Please comment out the `apt-get` before running the script.

```
$ ./setup.sh
```

Users can also manually install manifold with the following steps.

Step 1: Configuring Manifold

Inside your manifold directory, we now must configure the options for how Manifold will be build. The simplest way to do this is to just run

```
$ ./configure QSIMINC=$QSIM_PREFIX/include
```

`***Configuration Options***

For more control over the configuration script, the following options are available:

- `--prefix=PREFIX`
  By default, the header files and libraries will be installed in `/usr/local/include` and `/usr/local/lib`, respectively. If you want to install the files somewhere else, you should use this option, and the files will be installed in `PREFIX/include` and `PREFIX/lib`, respectively.

- `--disable-para-sim`
  By default, the Manifold libraries are built for parallel simulation with MPI. If you do not want to use MPI and therefore only run the simulators in sequential mode, you need to specify this option to disable parallel simulation.

- `--without-qsim`
  By default, when you build Manifold, you would have already built and installed QSim. If you do not want to use QSim, use this option when you run `configure`. When you use this option, the simulator can only use trace files.

- `--enable-kernel-large-data`
  By default, the maximum size of data that are sent between components is 1024 bytes. If this is not big enough, or the maximum size is not known in advance, then this option should be used.

- `--disable-stats`
  By default, the Manifold kernel and computer architecture models all collect statistics at run time. Use this option to disable run-time collection of statistics.

- `KERINC=KERNEL_LOCATION`
  This option specifies where the kernel header files are installed. This is useful when the kernel and the models are built separately.

- `QSIMINC=QSIM_LOCATION`
  This option specifies the location where QSim include files are installed. By default, QSim include files are installed under `/usr/local/include`. This option is useful when QSim is installed in a different location.

- `--enable-forecast-null`
  Use this option if you want to use the Forecast Null Message algorithm (FNM).

Step 2: Building Manifold

After all of the above configuration has been completed, building Manifold only requires the following commands:

```
$ make
$ make install (optional)
```

Note: “sudo” may be required.

Step 3: Download QSim Benchmarks

Currently the Splash-2 and graphBig benchmarks are prebuilt for manifold simulation. Please download benchmark tarballs using the links in setup.sh script.

Step 4: Copy the QSim State Files

Manifold simulation will use the QSim state files (i.e. state.n) to load the linux os image. Make sure the state files are prepared in advance. If there is no state files in the QSim directory, use the following script to generate the files in the QSim root directory.

```
$ cd QSIM_ROOT
$ ./mkstate.sh
```

Step 5: Building the Simulator Programs

The simulator programs are located in the following directory.

```
ROOT/simulator/smp
```

Simulators under the directory are highly configurable. You can configure the simulator components via the configure file in common directory. There are four subdirectories of simulators, based on how they get instructions:

- Programs under `QsimProxy` use proxy processes that work with QSim library to improve performance.
- Programs under `QsimClient` use QSim server to get instructions. To build these simulators, you must first build and install QSim.
- Programs under `QsimLib` use QSim libraries. To build these simulators, you must first build and install QSim.
- Programs under `TraceProc` use trace files created with a program based on Intel's PIN API.

In addition there are two other subdirectories:

- Subdirectory `common` contains code that is shared by all simulators.
- Subdirectory `config` contains configuration files that are shared by all simulators.

Currently, only QsimProxy simulator is supported. Use the following instructions to build:

```
$ cd ROOT/code/simulator/smp/QsimProxy
$ make
```

Note: modify `QsimProxy/Makefile` when necessary.

Run Manifold Simulation

Configure Manifold Simulator

The manifold configuration files are placed in the `smp/config` directory. There are 3 types configure files, as listed:

- **conf*.cfg** contains the configuration for manifold. Currently only spx configs are in use.
- **spx-*.config** contains the processor configuration for SPX core. The **`arch_type`** is specified in the file to choose between X86 and AArch64.
- **DDR2_*.ini** and **system.ini.example** are used for DRAMSim2 configuration.

In the QSIMProxy directory, there are two programs, **smp_llp** and **smp_l1l2**. The **smp_llp** simulates the following system model, where each core node has a processor core, a private L1 cache, and a shared L2 slice. The L1 and L2 cache are combined and connected to the interconnection through a MUX port. The LLP structure is a recommended configuration for multi-/many- core architecture simulation.

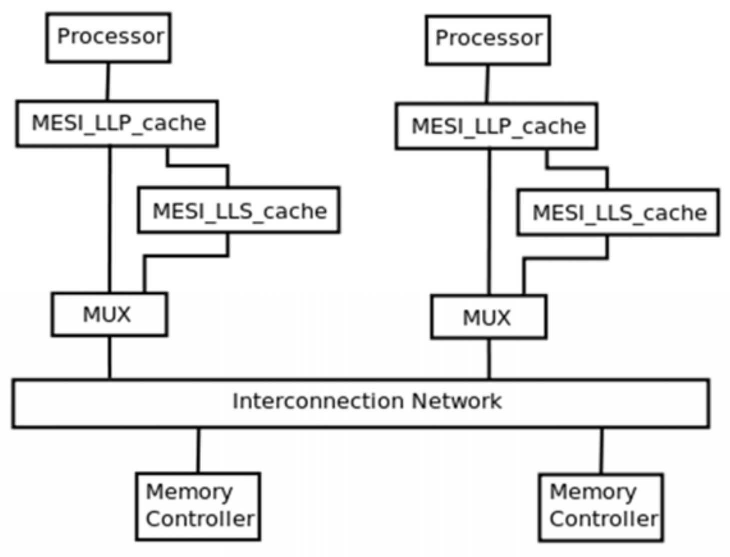

Figure 2. System Model used for smp_llp

The **smp_l1l2**, on the other hand, simulates a slightly different model where L1 and L2 nodes are separated between the networks.

Start the Simulation

The simulators under the QSimProxy directory can be run in both sequential and parallel mode with the following command lines:

```
$ mpirun -np <np> <prog> <conf_file> <state_file> <benchmark>
```

Where

- **<np>** is the number of logical processes (LPs), or MPI ranks. For parallel simulation, speficy np greater than 1. Simulator by default runs in sequential mode.
- **<prog>** is the simulator, including `smp_llp`, and `smp_l1l2`.
- **<conf_file>** is the configuration file for the system being simulated. The system configuration is defined in libconfig format.
- **<state_file>** is QSim's state file.
- **<benchmark>** is the application tar file.

Please check the QSIM_PREFIX variable to make sure QSim is installed correctly. An example command (sequential simulation) is shown as follow:

```
$ smp_llp ../config/conf2x3_spx_torus_llp.cfg ../state/state.4
../benchmark/myBench.tar
```

Appendix

I Common Problems

1. mpirun: command not found
   Solution: If you are using openmpi, install the openmpi-bin package.
2. simulation_stop has incorrect type.
   Solution: Open the configuration file, append an 'L' to the number you specify for simulation_stop. For example, if it was "simulation_stop = 1000", change it to "simulation_stop = 1000L".
3. cp: cannot stat `./libqemu.so': No such file or directory
   system("cp ./libqemu.so /tmp/qsim_WKLK7m") returned 256.
   Solution: Specify LD_LIBRARY_PATH as shown above.
4. cp: cannot stat `/usr/local/lib/libqemu-qsim.so': No such file or directory
   system("cp /usr/local/lib/libqemu-qsim.so /tmp/qsim_eIwV0x") returned 256.
   Solution: Specify QSIM_PREFIX as shown above.

II Configure the Sync Mechanism of Manifold Simulation

Manifold supports the following synchronization algorithms:

- SA_CMB: basic Null-message (CMB) algorithm.
- SA_CMB_OPT_TICK: optimized CMB for clock-cycle-based simulations. This is the default.
- SA_CMB_TICK_FORECAST: optimization of SA_CMB_OPT_TICK using Forecast Null-message (FNM).
- SA_LBTS: lower bound time-stamp.

SA_QUANTUM: time-quantum-based synchronization.

The default algorithm is SA_CMB_OPT_TICK. The algorithm can be set in the simulator program when calling the Manifold::Init() function.

For example, to set the algorithm to SA_CMB, do the following:

```
Manifold::Init(argc, argv, Manifold::TICKED, SyncAlg::SA_CMB);
```

The Quantum algorithm is slightly different. After calling Manifold::Init(), you need to call another function to set the quantum value. For example:

```
Manifold::Init(argc, argv, Manifold::TICKED, SyncAlg::SA_QUANTUM);
Quantum_Scheduler* sch = dynamic_cast<Quantum_Scheduler*>(Manifold::get_scheduler());
//get the scheduler
assert(sch);
sch->init_quantum(10); //set the quantum to 10 cycles
```

III Start Different Simulators

Start the Simulators in QSimClient

These simulators require QSim server be started first. To start the QSim server, run the following commands:

```
$ cd QSIM_ROOT/remote/server
$ make
$ ./qsim-server <port> <state_file> <benchmark>
```

where

- <port> is the TCP port number the server uses. Use any number you want.
- <state_file> is the state file. QSim is an emulator of a multicore shared-memory machine. The state file is the snapshot of the emulated machine after the OS has booted.
- <benchmark> is the tar file containing the application program and its data. See QSim instructions on how to build benchmark tar files.

After the QSim server has started, the simulator can be started.

If QSim is installed in `/usr/local`, do the following,

```
$ cd SIMULATOR_ROOT
$ mpirun -np <NP> <prog> <conf_file> <server> <port>
```

If Qsim is not installed in `/usr/local`, do the following, assuming QSim installation path is QSIM_INSTALL.

```
$ cd SIMULATOR_ROOT
$ QSIM_PREFIX=<QSIM_INSTALL> LD_LIBRARY_PATH=<QSIM_INSTALL>/lib  mpirun -np <NP>
<prog> <conf_file> <server> <port>
```

where

- <NP> is the number of logical processes (LPs), or MPI ranks. For parallel simulation, currently the simulators support 1, 2, or N+1 LPs, where N is the number of simulated cores.
- <prog> is the simulator, including `smp_llp`, and `smp_l1l2`.
- <conf_file> is the configuration file for the system being simulated. The system configuration is defined in libconfig format.
- <server> is the name or IP address of the QSim server.
- <port> is the TCP port number used by the QSim server.

For example:

```
$ mpirun -np 2 smp_llp ../config/conf2x2_torus_llp.cfg localhost 12345
```

The output of the simulation is stored in files named `DBG_LOG<i>`, where `<i>` is 0 to n-1, n being the number of LPs. The output files contain statistics collected by the components assigned to the corresponding LP.

Start the Simulators in QSimLib

Simulators in this subdirectory can only be run with 1 LP, or in sequential mode.

If QSim is installed in `/usr/local`, do the following.

```
$ mpirun -np 1 <prog> <conf_file> <state_file> <benchmark>
```

If Qsim is not installed in `/usr/local`, do the following, assuming QSim installation path is QSIM_INSTALL.

```
$ QSIM_PREFIX=<QSIM_INSTALL> LD_LIBRARY_PATH=<QSIM_INSTALL>/lib  mpirun -np 1 <prog>
<conf_file> <state_file> <benchmark>
```

where

- <prog> is the simulator, including `smp_llp`, and `smp_l1l2`.
- <conf_file> is the configuration file for the system being simulated. The system configuration is defined in libconfig format.

- <state_file> is QSim's state file.
- <benchmark> is the application tar file.

For example:

```
$ mpirun -np 1 smp_llp ../config/conf4x5_spx_torus_llp.cfg myState_16 myBench.tar
```

The output of the simulation is stored in a file named `DBG_LOG0`, which contains statistics collected by all of the components.

Start the Simulators in TraceProc

These simulators use traces obtained with a PIN-based program.

```
$ mpirun -np <NP> <prog> <conf_file> <trace_file_basename>
```

where

- <NP> is the number of logical processes (LPs), or MPI ranks. For parallel simulation, currently the simulators support 1, 2, or N+1 LPs, where N is the number of simulated cores.
- <prog> is the simulator, including `smp_llp`, and `smp_l1l2`.
- <conf_file> is the configuration file for the system being simulated. The system configuration is defined in libconfig format.
- <trace_file_basename> is the base name of the trace files. All trace files must have the same base name and be named <base_name>0, <base_name>1, <base_name>2, etc. For example, if the trace files are `myFile0`, `myFile1`, then the base name is `myFile`.

For example:

```
$ mpirun -np 2 smp_llp ../config/conf2x2_torus_llp.cfg myTrace
```

The output of the simulation is stored in files named `DBG_LOG<i>`, where `<i>` is 0 to n-1, n being the number of LPs. The output files contain statistics collected by the components assigned to the corresponding LP.

References

[1] QSim - QEMU for Simulators, www.cdkersey.com/qsim-web.

[2] Parsec Benchmark, http://parsec.cs.princeton.edu.

[3] Splash2 Benchmark, http://www.capsl.udel.edu/splash.

[4] GraphBig Benchmark, https://github.com/graphbig.

[5] QSim User Manual, http://manifold.gatech.edu/wp-content/uploads/2016/02/user_guide_2.2.pdf.

[6] Manifold Website, http://manifold.gatech.edu.