Manifold 0.12 User Guide

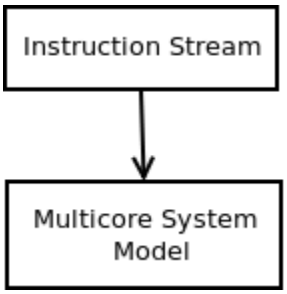
May 29, 2014

1 Introduction

Manifold is a parallel discrete event simulation framework for simulation of modern multicore computer architectures. The software mainly consists of two layers: a simulation kernel layer, and a model layer that contains a few computer architecture models. In addition, Manifold also provides a few ready-to-use simulator programs. This user guide describes how to obtain Manifold source code, and how to build and run the simulator programs.

2 Overview

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



At run-time, instruction streams are fed to the multicore system model for simulation. Example sources of instructions include PIN trace files and the QSim 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 model, cache model 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 Energy Introspector (EI) 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.

3 Current Release

The current release is Release 0.12. The software is distributed as a source code package that contains the following:

- The parallel simulation kernel.
- Models:
  - Zesto: a cycle-level x86 processor model.
  - SPX: a superscalar x86 processor model.
  - Simple-proc: a 1-IPC proxessor model.
  - 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.
- Simulator programs.

Zesto

- A cycle-level x86 processor model. This model can accept intruction 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. It can accept instruction streams from QSim server or QSim library.

Simple-proc

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.

3.1 Testing and Portability

- Manifold has been tested on the PARSEC and SPLASH-2 benchmarks.
- Manifold has been tested under the the following operating systems:
  - Ubuntu 10.04, 12.04.
  - Fedora release 17.
- Manifold has been tested with Openmpi and MPICH2.

4 Directory Structure

The Manifold source code is organized as follows:

```
ROOT
|... code
     |... doc
     |... kernel
     |... models
     |    |... cache
     |    |    |... mcp-cache
     |    |    |... simple-cache
     |    |... memory
     |    |    |... CaffDRAM
     |    |... network
     |    |    |... iris
     |    |... processor
     |         |... simple-proc
     |         |... zesto
     |... simulator
          |... smp
          |    |... common
          |    |... config
          |    |... QsimClient
          |    |... QsimLib
          |    |... QsimProxy
          |    |... TraceProc
          |... smp2
               |... common
               |... config
               |... QsimClient
               |... QsimLib
               |... TraceProc
```

where *ROOT* represents the root of the source tree.

The simulator directory

There are two sets of simulator programs, under `smp` and `smp2`, respectively. Programs under `smp` use a fixed set of components, while programs under `smp2` allows one component to be replaced by another by simply changing a configure file. For example, the user can replace Zesto with SPX.

Under each of `smp` and `smp2`, there are a few subdirectories: `QsimClient`, `QsimLib`, `QsimProxy`, and `TraceProc`. The difference is the source of instruction streams. Programs under `QsimClient` and `QsimProxy` use QSim server to get instructions. Those under `QsimLib` are built

with the QSim libraries. And those under `TraceProc` use trace files.

The common code of the simulator programs is located in `common`. The directory `config` contains configuration files for the simulator programs.

5 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.
- Build the simulator program(s).
- Run the simulators.

The simulators can respectively take instructions from three different sources: trace files, QSim library, and QSim server. Depending on which source you use, some of the steps above may be optional.

The following explains each step in detail.

6 Install Required Packages

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

- mpi: We have tested with openmpi, so it is recommended.
- libconfig++: The simulators require this package.

7 Download and Build QSim

If you choose to use QSim to get instructions, you need to build and install QSim first.

7.1 Download

We recommend using QSim version 0.1.5, which is available at the Manifold web site:

- http://manifold.gatech.edu/download

7.2 Build and Installation

Instructions for building and installing QSim can be found in the `INSTALL` file in the root directory of

QSim source code.

In addition to the QSim libaries, you also need to do the following:

- build the QSim server.
- build and install the QSim client library.

All the instructions are in the `INSTALL` file.

After you are finished, you installation directory should look like the following, assuming QSIM_INSTALL is the root of the installation directory.

```
$ ls <QSIM_INSTALL>/lib
libqemu-qsim.so  libqsim-client.so  libqsim.so

$ ls <QSIM_INSTALL>/include
mgzd.h  qsim-client.h  qsim.h  qsim-load.h  qsim-net.h  qsim-regs.h  qsim-vm.h
```

8 Download and Build Manifold Libraries

There are two ways to download Manifold source code: from the Manifold website or through SVN checkout. Depending on which way is used to obtain the source code, the build process is slightly different.

8.1 Download Manifold source package

Manifold source package is available at the Manifold website:

- http://manifold.gatech.edu/download

After download, follow the following instructions to build the manifold libraries:

1. Untar the source package.

   ```
   $ tar xvfz manifold-0.11.1.tar.gz
   ```

2. Go to the `code` subdirectory.

   ```
   $ cd manifold-0.11.1/code
   ```

3. Run `configure` and `make`.

   ```
   $ ./configure [--prefix=INSTALL_PATH]
   $ make
   ```

   The default installation directory is `/usr/local/lib`. If you want to install in a different location, the path of that location should be passed to `configure`. In addition, if QSim is installed in a location other than the default, you need to tell `configure` that location.

Options that you can specify for `configure` are described below.

4. Optionally, install the libraries.

   ```
   $ make install
   ```

8.2 Download Manifold source code through SVN checkout

Manifold source code is available through SVN checkout at the following address:

- https://svn.ece.gatech.edu/repos/Manifold/tags/0.11.1

To build the un-packaged source code, you need to have autotools package installed on your machine.

1. From the `code` subdirectory, run `autoreconf`.

   ```
   $ cd code
   $ ./autoreconf -si
   ```

   This would create the `configure` script.

2. Run `configure` and `make`.

   ```
   $ ./configure [--prefix=INSTALL_PATH]
   $ make
   ```

3. Optionally, install the libraries.

   ```
   $ make install
   ```

8.3 Configure options

This section describes all of the options you can use when running the `configure` script.

- `--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 is installed. By default, QSim is installed under `/usr/local`. 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).

9 Build the Simulator Program

The simulator programs are located in `ROOT/code/simulator/smp` and `ROOT/code/simulator/smp2`. Programs under `smp` use a fixed set of components: Zesto, MCP-cache, Iris, and CaffDRAM. Those under `smp2` are more flexible. They allow a component to be replaced by another by simply modifying the configure file. For example, you can replace Zesto with SimpleProc, or CaffDRAM with DRAMSim2.

There are four subdirectories of simulators, based on how they get instructions:

- 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 `QsimProxy` use proxy processes that work with QSim server to improve performance.
- 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.

To build the simulators, follow the following steps. Here we use the simulators under `QsimClient` as an example.

1. Go to the simulator source directory.

```
$ cd ROOT/code/simulator/zesto/QsimClient
```

2. Run `make`. It is likely that you need to modify the Makefile so the header files and libraries can be found.

   ```
   $ make
   ```

10 Start the Simulator

In each of the subdirectories, there is a program called **smp_`llp`**. This program simulates the following system model, where each core node has a processor core, a private L1 cache, and a shared L2 slice.

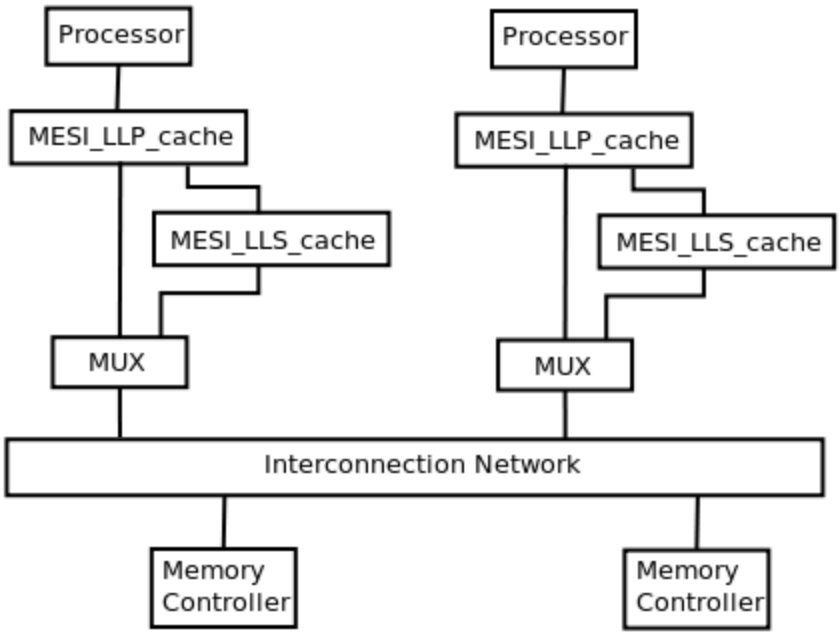


Figure 1 System Model Simulated by smp_llp.

In addition, there is a program called **`smp_l1l2`**. It simulates a slightly different model in which L2's are in separate nodes, like memory controllers.

We describe how to start the simulators in each of the subdirectories.

10.1 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 zesto_llp, and zesto_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
```

10.2 Start the Simulators in QsimProxy

Simulators in `QsimProxy` use proxy processes that is placed between the QSim server and the back-end timing simulation to improve performance. The proxy processes act as client to the QSim server and obtains instructions from the QSim server over TCP/IP. The proxies and the back-end simulation form a producer-consumer relationship using shared memory segments. The proxies put instructions in the shared memory segments to be removed by the back-end simulation. The proxies keeps monitoring the contents of the shared memory segments. Once the contents fall below a threshold, they communicate with the server to obtain more instructions.

The processes should be started in the following order.

1. Start the QSim server.
2. Start the proxy processes.
3. Start the simulator.

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 proxies are started. Each proxy is a multithreaded process that can serve multiple core models in the back-end. Obviously, the proxy and the core models they serve must run on the same physical machine because they communication with shared memory segments.

```
$ cd ROOT/models/processor/zesto/proxy_mt
$ make
$ ./proxy <qsim_server> <port> <shared_mem_filename> <shared_mem_size>
<core-proxy_map>
```

where

- <qsim_server> is the host name or IP address of the QSim server.
- <port> is the TCP port number the server uses.
- <shared_mem_filename> is a file accessible by all your processes.
- <shared_mem_size> is size of each shared memory segment. There's one segment for each core model. The size must be a power of 2.

- <core-proxy_map> is a mapping file that maps each core id to the host name of its proxy.

An example core-proxy map file is as follows:

```
0  crankshaft
1  crankshaft
2  crankshaft
...
15 crankshaft
```

This file is for a simulation model that has 16 cores. All the cores are served by a single proxy running on the machine `crankshaft`.

After the QSim server and the proxies have started, the simulator can be started.

```
$ cd SIMULATOR_ROOT
$ mpirun -np <NP> <prog> <conf_file> <shared_mem_filename> <shared_mem_size>
```

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.
- <shared_mem_filename> is a file accessible by all your processes.
- <shared_mem_size> is size of each shared memory segment. There's one segment for each core model. The size must be a power of 2.

For example:

First start the proxy:

```
$ ./proxy localhost 12345 ~/shm_file 262144 ./core_proxy_map
```

Then start simulator:

```
$ mpirun -np 9 smp_llp ../config/conf4x5_torus_llp.cfg ~/shm_file 262144
```

Unlike the other simulator programs, with proxies, we put two core models in a single MPI process (LP). Therefore, for a 16-core model, we use 9 processes (8 for the cores, 1 for network and memory controllers).

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.

10.3 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 `zesto_llp`, and `zesto_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/conf4x1_ring_llp.cfg myState_16 myBench.tar
```

10.4 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 `zesto_llp`, and `zesto_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
```

10.5 Selecting Synchronization Algorithm

Manifold supports the following synchronization algorithms:

- `SA_CMB:` basic Null-messsage (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
```

11 Common Problems

The following is a list of commonly encountered problems, and how to solve them.

- mpirun: command not found
  Solution: If you are using openmpi, install the openmpi-bin package.
- 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".

- 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.
- 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.