Extensible Flit-Level Simulation of Large-Scale Interconnection Networks

Nic McDonald, Adriana Flores, Al Davis, Mikhail Isaev, John Kim, Doug Gibson
Why Reinvent the Wheel?
SuperSim: Features and Attributes

- Fast event-driven simulation
  - Only model things that change

- Single threaded (really is this a feature?)
  - Easy to use “run to completion” of each event.
  - Simulations achieve 50k to 5M events per second.

- Source code:
  - ~40,000 lines of code
  - ~400 source/header files
  - 10+ external libraries
  - Supported by many tools

“If a simulator already does what you want it to do, you're most likely asking the wrong questions.”
- Professor Christos Kozyrakis (Stanford CS/EE)
Settings and Configuration

– Extended JSON to configure a simulation
– Command line configuration modifiers
– Hierarchical nature of JSON matches the hierarchical structure of simulation:

```
$ supersim myconfig.json \
> network.router.architecture=string=output_queued \
> network.levels=uint=4
```

```
"network": {
  "topology": "folded_clos",
  "levels": 3,
  "radix": 6,
  "protocol_classes": [{
    "num_vcs": 2,
    "routing": {
      "algorithm": "common_ancestor",
      "latency": 1, // cycles
      "least_common_ancestor": true,
      "mode": "port",
      "adaptive": false
    }
  }],
  "router": {
    "architecture": "input_queued",
    "input_queue_depth": 100,
    "output_queue_depth": 164,
    "crossbar": {
      "latency": 8 // cycles
    },
    "vc_scheduler": {
      "allocator": {
        "type": "rc_separable",
        "slip_latch": true,
        "iterations": 2,
        "resource_arbiter": {
          "type": "lslp"
        },
        "client_arbiter": {
          "type": "lslp"
        }
      }
    }
  }
}
```
Smart Object Factories

- Factories with zero-modify module inclusion
- New model files can just be dropped in.
- No code changes required to the code base.

```cpp
#include "traffic/continuous/LoopbackCTP.h"
#include <factory/Factory.h>

LoopbackCTP::LoopbackCTP(
    const std::string& _name, const Component* _parent,
    u32 _numTerminals, u32 _self, Json::Value _settings)
    : ContinuousTrafficPattern(
        _name, _parent, _numTerminals, _self, _settings) {}

LoopbackCTP::~LoopbackCTP() {}

u32 LoopbackCTP::nextDestination() {
    return self_;}

registerWithFactory(
    "loopback", ContinuousTrafficPattern,
    LoopbackCTP, CONTINUOUSTRAFFICPATTERN_ARGS);
```
Use realistic architectural models

- Router pipelines
- Routing algorithms
- Credit management
- Congestion detection
SuperSim Structure
Simulator Core

Simulator
Priority Queue
Executor

Component A
Component B
Component C

Tick 0
Eps 0 Eps 1 Eps N
Tick 1
Eps 0 Eps 1 Eps N Eps 0

Ticks
Epsilons

Clock A
Clock B

Tick 0
Eps 0 | Eps 1 | • • • | Eps N
Tick 1
Eps 0 | Eps 1 | • • • | Eps N | Eps 0

Time
Simulator Architecture

[Diagram showing the architecture with Application, Workload, and Terminal nodes connected through Interface and Router nodes.]

[Time axis showing states like Warming, Generating, Finishing, Draining, Ready, Start, Complete, Stop, Done, Kill, and Sampling.]
Multi-Application Workload Example

- Explore transient analysis of adaptive routing
- “Blast” application running steady state traffic
- “Pulse” application generates a temporary disturbance with a batch of traffic
Network Topologies (not trying to cover the whole space)

Real Topologies
- Torus
  - Oblivious routing
- Folded-Clos
  - Oblivious and adaptive routing
- HyperX
  - Can generate all HyperCubes and Flattened Butterflies
  - Oblivious and adaptive (to be released soon) routing
- Dragonfly
  - Oblivious and adaptive routing
- SlimFly (to be released soon)
  - Oblivious and adaptive routing

Testing Topologies
- Uno
  - A single router
- ParkingLot
  - A cascade of routers to stress bandwidth fairness
Router Architectures (definitely not covering the whole space)
VCs, RCs, PCs, and TCs

- Traffic Classes (TCs)
- Protocol Classes (PCs)
- Routing Classes (RCs)
- Virtual Channels (VCs)

| VC 0 | Routing Class 0 – VDAL hop 0 |
| VC 1 | Routing Class 1 – VDAL hop 1 |
| VC 2 | Routing Class 2 – VDAL hop 2 |
| VC 3 | Routing Class 3 – VDAL hop 3 |
| VC 4 | Routing Class 4 – VDAL hop 4 |
| VC 5 | Routing Class 5 – VDAL hop 5 |
| VC 6 | Routing Class 6 – VDAL hop 0 |
| VC 7 | Routing Class 7 – VDAL hop 1 |
| VC 8 | Routing Class 8 – VDAL hop 2 |
| VC 9 | Routing Class 9 – VDAL hop 3 |
| VC 10 | Routing Class 10 – VDAL hop 4 |
| VC 11 | Routing Class 11 – VDAL hop 5 |
| VC 12 | Routing Class 12 – DOR |
| VC 13 | Routing Class 13 – DOR |
| VC 14 | Routing Class 14 – DOAL min hops |
| VC 15 | Routing Class 15 – DOAL deroute hops |
| VC 16 | Routing Class 16 – DOAL min hops |
| VC 17 | Routing Class 17 – DOAL deroute hops |

Traffic Class 0

Traffic Class 1

Traffic Class 2

Traffic Class 3

Traffic Class 4

Traffic Class 5
Simulation Experiments
Latent Congestion Sensing

High-radix problem – many input ports bombard a seemingly good output port

Congestion latency – the time it takes for the input ports to see the congestion changes on the output ports

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network topology</td>
<td>3-level folded-Clos, 4096 terminals</td>
</tr>
<tr>
<td>Network channel latency</td>
<td>50 ns (i.e., 10 meter cables)</td>
</tr>
<tr>
<td>Routing algorithm</td>
<td>adaptive uprouting</td>
</tr>
<tr>
<td>Router radix</td>
<td>32 ports</td>
</tr>
<tr>
<td>Router architecture</td>
<td>output-queued (OQ)</td>
</tr>
<tr>
<td>Frequency speedup</td>
<td>1x (i.e., none)</td>
</tr>
<tr>
<td>Number of VCs</td>
<td>1 VC</td>
</tr>
<tr>
<td>Input buffer size</td>
<td>150 flits</td>
</tr>
<tr>
<td>Output buffer size</td>
<td>infinite or 64 flits</td>
</tr>
<tr>
<td>Router core latency</td>
<td>50 ns queue-to-queue</td>
</tr>
<tr>
<td>Message size</td>
<td>1 flit</td>
</tr>
<tr>
<td>Traffic pattern</td>
<td>uniform random to root</td>
</tr>
</tbody>
</table>

Hewlett Packard Enterprise
Congestion Credit Accounting

– Use UGAL routing, test different credit account mechanisms:
  – Output, downstream, output-and-downstream
  – VC, port

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network topology</td>
<td>1D flattened butterfly, 32 routers, 1024 terminals</td>
</tr>
<tr>
<td>Network channel latency</td>
<td>50 ns (i.e., 10 meter cables)</td>
</tr>
<tr>
<td>Routing algorithm</td>
<td>UGAL</td>
</tr>
<tr>
<td>Router radix</td>
<td>63 ports</td>
</tr>
<tr>
<td>Router architecture</td>
<td>input-output-queued (IOQ)</td>
</tr>
<tr>
<td>Frequency speedup</td>
<td>2x</td>
</tr>
<tr>
<td>Number of VCs</td>
<td>2 VCs</td>
</tr>
<tr>
<td>Input buffer size</td>
<td>128 flits</td>
</tr>
<tr>
<td>Output buffer size</td>
<td>256 flits</td>
</tr>
<tr>
<td>Router core latency</td>
<td>50 ns main crossbar latency</td>
</tr>
<tr>
<td>Message size</td>
<td>1 flit</td>
</tr>
<tr>
<td>Traffic pattern</td>
<td>uniform random, bit complement</td>
</tr>
</tbody>
</table>
Flow Control Techniques

- Flit-buffer flow control (FB)
- Packet-buffer flow control (PB)
- Winner-take-all flow control (WTA)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network topology</td>
<td>4D torus 8x8x8x8, 4096 terminals</td>
</tr>
<tr>
<td>Network channel latency</td>
<td>5 ns (i.e., 1 meter cables)</td>
</tr>
<tr>
<td>Routing algorithm</td>
<td>dimension order routing</td>
</tr>
<tr>
<td>Router radix</td>
<td>9 ports</td>
</tr>
<tr>
<td>Router architecture</td>
<td>input-queued (IQ)</td>
</tr>
<tr>
<td>Frequency speedup</td>
<td>1x (i.e., none)</td>
</tr>
<tr>
<td>Number of VCs</td>
<td>2, 4, 8 VCs</td>
</tr>
<tr>
<td>Input buffer size</td>
<td>128 flits</td>
</tr>
<tr>
<td>Output buffer size</td>
<td>n/a</td>
</tr>
<tr>
<td>Router core latency</td>
<td>25 ns main crossbar latency</td>
</tr>
<tr>
<td>Message size</td>
<td>1, 2, 4, 8, 16, 32 flits</td>
</tr>
<tr>
<td>Traffic pattern</td>
<td>uniform random</td>
</tr>
</tbody>
</table>

![Throughput vs Message Size](chart1.png)

![Throughput vs Message Size](chart2.png)

![Mean Latency vs Load](chart3.png)
Accompanying Tools
Simulation Pipeline

1. Configure
   – Create the simulation configurations needed for the experiment

2. Simulate
   – Run the simulations using the configurations

3. Parse
   – Parse the results of the simulation outputs into the format needed in the remaining steps

4. Analyze
   – Analyze the parsed results from simulation to create desired statistics

5. Plot
   – Generate plots of analysis data

6. View
   – View the analyzed and plotted results

This can easily turn into 10s of 1000s of command line operations!!!
Taskrun

– A Python package for declaring tasks and automated execution
– Generic task API supports:
  – Function tasks – executed as a Python function callback
  – Process tasks – locally executed command
  – Cluster tasks – a remotely executed command via a cluster scheduler (e.g., PBS, LSF, Slurm, etc.)
  – Your next big idea…
– Resource management (e.g., memory, CPUs, etc.)
– Dependencies and conditional execution (i.e., like a Makefile)

* Not a tool specific to SuperSim
SSParse

- SuperSim outputs a file containing information for all traffic from the “sampling” window (e.g., *.mpf).
- SSParse parses this file, run analyses, and prepares data sets for plotting
- SSParse exposes a filtering API to only view the information you care about
  - Ex: “+app=1” – only parses data from application 1
  - Ex: “-send=450-890” – parses data not sent between time 450 and 890
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th># of Sims</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time-Latency-Scatter</td>
<td>Load vs. latency scatter</td>
<td>1</td>
</tr>
<tr>
<td>Latency-PDF</td>
<td>Latency probability density function</td>
<td>1</td>
</tr>
<tr>
<td>Latency-CDF</td>
<td>Latency cumulative distribution function</td>
<td>1</td>
</tr>
<tr>
<td>Latency-Percentile</td>
<td>Latency percentiles (inverted logarithmic CDF)</td>
<td>1</td>
</tr>
<tr>
<td>Time-Latency</td>
<td>Time vs. latency at all distributions</td>
<td>1</td>
</tr>
<tr>
<td>Time-Average-Hops</td>
<td>Time vs. average hops</td>
<td>1</td>
</tr>
<tr>
<td>Time-Percent-Minimal</td>
<td>Time vs. minimal and non-minimal percentages</td>
<td>1</td>
</tr>
<tr>
<td>Load-Latency</td>
<td>Load vs. latency at all distributions</td>
<td>1 sweep</td>
</tr>
<tr>
<td>Load-Rate</td>
<td>Offered rate vs. delivered rate (min, mean, max)</td>
<td>1 sweep</td>
</tr>
<tr>
<td>Load-Rate-Percent</td>
<td>Offered rate vs. delivered rate (total, minimal, non-minimal)</td>
<td>1 sweep</td>
</tr>
<tr>
<td>Load-Average-Hops</td>
<td>Load vs. average hops</td>
<td>1 sweep</td>
</tr>
<tr>
<td>Load-Percent-Minimal</td>
<td>Load vs. minimal and non-minimal percentages</td>
<td>1 sweep</td>
</tr>
<tr>
<td>Load-Latency-Compare</td>
<td>Load vs. latency across multiple sweeps</td>
<td>N sweeps</td>
</tr>
</tbody>
</table>
SSPlot: Time-Latency-Scatter

Load vs. latency scatter
– This is the result of a single simulation
SSPlot: Latency-PDF

Latency probability density function

– This is the result of a single simulation
SSPlot: Latency-CDF

Latency cumulative distribution function

– This is the result of a single simulation
Latency percentiles (inverted logarithmic CDF)
– This is the result of a single simulation
SSPlot: Time-Average-Hops

**Time vs. average hops**

- This is the result of a single simulation

- In this simulation there are two applications. Application 0 sends uniform random traffic at 10% the whole time. Application 1 sends bit complement traffic (adversarial) in a pulse starting at time ~35,000

- These results show the traffic only for Application 0
SSPlot: Time-Percent-Minimal

Time vs. minimal and non-minimal percentages

– This is the result of a single simulation

– In this simulation there are two applications. Application 0 sends uniform random traffic at 10% the whole time. Application 1 sends bit complement traffic (adversarial) in a pulse starting at time ~35,000

– These results show the traffic only for Application 0
SSPlot: Load-Latency

Load vs. latency at all distributions

- This is the result of a sweep of simulations across injection rate
- This simulation is an application sending uniform random traffic and randomly sizes messages
SSPlot: Load-Latency-Compare

Load vs. latency across multiple sweeps

- This is the result of many sweeps of simulations across injection rate (one sweep for “RR” and one for “AGE”)

- This plot is like the Load-Latency plot but compares across multiple sweeps

- This particular setup shows median latency (any latency distribution can be chosen)
SSPlot: Load-Rate

Offered rate vs. delivered rate (min, mean, max)

– This is the result of a sweep of simulations across injection rate

– This simulation is an application sending traffic over a torus network that stresses the bisection

– At 65% injection rate, the network becomes saturated. Severe bandwidth unfairness occurs due to round-robin arbitration
SSSweep

– SSSweep automates the entire simulation pipeline process
– Users define independent simulation variables and corresponding functions to apply the variable

```
alg = ['oblivious', 'adaptive']

def set_alg(alg, config):
    return ('network.protocol_classes[0].routing.adaptive=bool={}'
            .format('true' if alg == 'adaptive' else 'false'))

sweeper.add_variable('Routing Algorithm', 'RA', alg, set_alg)
```

– Users define the type of plots they’d like
– SSSweep creates all configurations and uses Taskrun to run all tasks
– SSSweep generates a static HTML/CSS/Javascript web site for plot viewing
SuperSim: Extensible Flit-Level Simulation of Large-Scale Interconnection Networks

www.github.com/hewlettpackard/supersim
Focus on Real Issues of Large-Scale Networks

Analyze latency distributions rather than just average latency.
SSParse: Transient Tool

- SSParse includes a wrapper tool that uses the main SSParse executable to generate a transient analysis.
SSPlot: Time-Latency

Time vs. latency at all distributions

– This is the result of a single simulation

– In this simulation there are two applications. Application 0 sends uniform random traffic at 10% the whole time. Application 1 sends bit complement traffic (adversarial) in a pulse starting at time ~35,000

– These results show the traffic only for Application 0
SSPlot: Load-Rate-Percent

Offered rate vs. delivered rate (total, minimal, non-minimal)

– This is the result of a sweep of simulations across injection rate
– This simulation is an application sending uniform random traffic and randomly sizes messages
SSPlot: Load-Average-Hops

Load vs. average hops

– This is the result of a sweep of simulations across injection rate
– This simulation is an application sending uniform random traffic and randomly sizes messages
SSPlot: Load-Percent-Minimal

Load vs. minimal and non-minimal percentages

– This is the result of a sweep of simulations across injection rate

– This simulation is an application sending uniform random traffic and randomly sizes messages