GVSOC simulator

Germain Haugou  haugoug@iis.ee.ethz.ch
Nazareno Bruschi  nazareno.bruschi@unibo.it
Why another simulator?

All-in-one simulator for Pulp chips

- SW development and debug
- Timing and power analysis
  - Benchmarking and profiling
  - Architecture exploration

Trade-off between simulation speed and accuracy:

- 20 mips with timing shared between simulated cores
- 100 mips without timing
- Timing under 10% of error for nominal cases, 20% for corner cases
- Power under 20% of error
Models

Component-based
C++
Lifecycle (build, start, stop, etc)
Signatures for interfaces
Set of methods for each signature
Callbacks for remote calls
Descriptor in Python for composition

```python
class Memory():
  def __init__(self, parent, name, size):
    self.set_model("memory")
    self.add_property("size", size);

void Memory::build()  
{  
  new_master_port("irq", &irq_itf);
  size = get_property("size");
}

void Memory::start() 
{  
  irq_itf.sync(1);
}
```
Generators

Python scripts to describe system hierarchy

Component-based

Instantiate, configure and bind components

Suitable for quick architecture exploration

User properties for command-line customization

```python
class Cluster():
    def __init__(self, parent, name):
        l1 = Memory(size=16*1024)
        pe0 = Iss(isa=rv32imfc)
        self.bind(pe0, "data", l1, "input"
        ...

    self.add_user_property("size", 16*1024)
    l1 = Memory(size=self.get_user_property("size")

--target-property chip/cluster/l1/size=32*1024
```
Full system simulation

Support for full systems with chips and devices

Devices are available: flash, rams, cameras, audio devices.

Based on generators

Generator reuse for complex system, e.g. a multi-board system

class Gap9_evk(tree.Component):
    def __init__(self, parent, name):
        Chip = Gap9()
        Flash = Hyperflash(size=8*1024*1024)
        self.bind(chip, "hyper0", flash, "input")

board1 = Gap9_evk()
board2 = Gap9_evk()
self.bind(board1, "uart0", board2, "uart1")
Simulation engine

Event-based simulation

Seen as cycle-based from SW

Callbacks executed at specific cycles

Clock engines for scheduling callbacks in frequency domains

Global time engine to schedule clock engines

Stubs to synchronize cycles between clock engines
Functional modeling

Target is 100% equivalence with real system

A binary can run unmodified on simulator and real system

Model can be bit-accurate if needed

Clock usually not modeled

Everything visible from SW is modeled

Some models can be fully accurate, e.g. padframe interface, or fll.
Timing modeling

All models are timed: instructions, memory accesses, DMAs, contentions, register accesses, etc.

Callbacks scheduled at specific cycles to reproduce expected timing
Memory-mapped requests

Synchronous

Go through all components in same cycle

Latency immediately reported

Asynchronous

Go through all components in several cycles

Any component can delay the grant or the response

Latency created with callback scheduling

Masters must supports both behaviors

Any slave can switch from synchronous to asynchronous
Frequency scaling

Models manipulate cycles

- Automatic frequency scaling
- Event remains scheduled for same cycles
- Clock engine position in time engine automatically updated
- Stub automatically convert cycles when crossing domains

Models can manipulate several clock engines
Power models

- Models can add power sources
  - Background power for leakage or background activity (e.g. idle)
  - Energy quantum to assign a cost to an event (e.g. and instruction).
  - Power values interpolated from calibration tables
  - Interpolated from current temperature, voltage and frequency

- Hierarchical report for average power

- GUI for Instantaneous power
GVSOC API and cosimulation

GVSOC C++ API for external control

To instantiate GVSOC and control its execution

To synchronize timing

Used for cosimulation, to use device models for HW verification.

- Device models and tests can be used to verify a chip and its model
GTKWave support

VCD (or FST) traces generated by models events (instructions, DMA transfers, etc)

Gtkwave script generated to organize signals

Limited to post-processing and small number of events.
GUI

On-going work to replace gtkwave and support huge number of events.

Events stored in a database with pre-computed averaging at multiple levels for fast queries.

Swap mechanism with lz4 compression for better footprint.

Handle smoothly hours of simulations with billions of events
Linux support

The ISS supports rv64imafdc and MMU

It can boot Linux (until console entry) in 55s at 70Mips

On-going work to bring it to 120Mips without timing.

Will be connected soon to pulp cluster
GDB support

GDB server for gdb connection

Each core is seen as a thread

All gdb commands available

<table>
<thead>
<tr>
<th>Id</th>
<th>Target Id</th>
<th>Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>*1</td>
<td>Thread 1 (pe0)</td>
<td>eu_evt_maskWaitAndClr (evtMask=4) at archi/chips/gap9_v2/event_unit/event_unit.h:173</td>
</tr>
<tr>
<td>2</td>
<td>Thread 2 (pe1)</td>
<td>eu_evt_maskClr (evtMask=4) at archi/chips/gap9_v2/event_unit/event_unit.h:132</td>
</tr>
<tr>
<td>3</td>
<td>Thread 3 (pe2)</td>
<td>eu_evt_maskClr (evtMask=4) at archi/chips/gap9_v2/event_unit/event_unit.h:132</td>
</tr>
<tr>
<td>4</td>
<td>Thread 4 (pe3)</td>
<td>eu_evt_maskClr (evtMask=4) at archi/chips/gap9_v2/event_unit/event_unit.h:132</td>
</tr>
<tr>
<td>5</td>
<td>Thread 5 (pe4)</td>
<td>eu_evt_maskClr (evtMask=4) at archi/chips/gap9_v2/event_unit/event_unit.h:132</td>
</tr>
<tr>
<td>6</td>
<td>Thread 6 (pe5)</td>
<td>eu_evt_maskClr (evtMask=4) at archi/chips/gap9_v2/event_unit/event_unit.h:132</td>
</tr>
<tr>
<td>7</td>
<td>Thread 7 (pe6)</td>
<td>eu_evt_maskClr (evtMask=4) at archi/chips/gap9_v2/event_unit/event_unit.h:132</td>
</tr>
<tr>
<td>8</td>
<td>Thread 8 (pe7)</td>
<td>0x1c010738 in __udivmoddi4 (rp=&lt;synthetic pointer&gt;, d=&lt;optimized out&gt;, n=7) at riscv-gnu-toolchain/riscv-gcc/libgcc/libgcc2.c:1077</td>
</tr>
<tr>
<td>9</td>
<td>Thread 9 (pe8)</td>
<td>pi_cl_team_nb_cores () at kernel/chips/gap9/cluster/cluster_task.h:65</td>
</tr>
<tr>
<td>10</td>
<td>Thread 10 (mchan)</td>
<td>0x00000000 in ?? ()</td>
</tr>
<tr>
<td>11</td>
<td>Thread 11 (fc)</td>
<td>__pi_task_sleep_loop () at kernel/task_asm.S:62</td>
</tr>
</tbody>
</table>
What can we do with these features?

WiPLASH EU project case

- Analyze system bottlenecks in AI applications: computation wall vs memory wall
  1. Add accelerators to speed-up the execution
  2. Increase the number of cores
  3. Execute end-to-end networks

- Proposal
  1. Analog In-Memory Computing based accelerator
  2. Heterogenous (digital-analog) multi-cluster PULP architecture configuration
  3. Real-life CNN inference

- We need a simulator to quickly get accurate timing results
  - Extend GVSOC to support analog accelerator, multi-cluster architecture and interconnect subsystem
Background
Analog In-Memory Accelerator

- Fixed Computation Latency (Analog)
- 256x256 Crossbar Size
- 64 byte/cycle Streamers Bandwidth
- Digital (data streams) and analog (MVM) tasks can be overlapped
# include <ima.hpp>

// declare the register map
int32_t reg_map = [IMA_REG_MAP_SIZE]

void access_ima_reg_map(req)
{
    // extract info from incoming req
    offset = req->offset;
    if req->is_write == true
        reg_map[offset] = req->data;
    else
        req->data = reg_map[offset];
    // trigger controller
    if offset == IMA_TRIGGER
        ima_trigger();
}
Cluster

- 1 IMA
- 16 RISC-V cores (PULP extensions)
- 1 MB of L1 Memory (Tightly-Couple Data Memory)
- 1 DMA
- Cores, DMA and IMA tasks can be overlapped
def build_cluster(clust, config):
    # configuration file in JSON format
    info = extract_info(config)
    # declare RISCV cores
    for id in range(info.nb_cores):
        cores[id] = RV_CORE(info.core)
    # declare TCDM
    tcdm = TCDM(info.tcdm)
    # declare accelerator
    ima = IMA(info.ima)
    # declare crossbar
    xbar = XBAR(info.xbar)
    # bind components
    clust.bind(ima.master, xbar.slave)
clust.bind(xbar.master, tcdm.slave)
...
• 512 clusters (CL) @ 1 GHz
• On-Chip Hierarchical Interconnect
  ▪ 4 interconnect levels (L1, L2, L3, Wrapper)
  ▪ 512 Gbit/s bandwidth each
  ▪ 4 cycles latency each
• Off-Chip HBM link 512 Gbit/s, 100 cycles latency
Architecture declaration

JSON files

```
“system_interconnect”: {
  “type”: “hierarchical”,
  “structure”: {
    “L1”: {
      “nb_masters”: 4,
      “nb_slaves”: 4,
      “model”: “axi_xbar”,
      “bandwidth”: 64,
      “latency”: 4
    },
    …
  }
}
```
Use case

PULP Anniversary 4-6 June 2023 Lugano, Switzerland
Execution model

Pipeline stages

0
conv

in
compute
out

1
pool

in
compute
out

2
conv

in
compute
out

batch ID

\( \rightarrow \) one new batch after \( \text{MAX(in,compute,out)} \)

<table>
<thead>
<tr>
<th>Batch ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results

![Graph showing latency by cluster ID with marked categories: EXEC (ANALOG BOUND), EXEC (DIGITAL BOUND), SYNCHRONIZATION OVERHEAD, COMMUNICATION OVERHEAD, and IDLE.]

- EXEC (ANALOG BOUND)
- EXEC (DIGITAL BOUND)
- SYNCHRONIZATION OVERHEAD
- COMMUNICATION OVERHEAD
- IDLE

Latency [ms] vs. Cluster ID
Consideration
Future work

Power modeling
  Full chip support (Gap) with associated RAM and flash

GVSOC API
  Better interoperability with other simulators
  Host emulation

More Pulp models (Snitch, Occamy, etc), Linux connection

Model contributions
  Clear APIs
  Documentation
Thank you!