Developing Dynamic Resource Management System in SoC Emulation

Seonchang Choi, Sangwoo Noh, Seonghee Yim, Seonil Brian Choi
Index

- Backgrounds
- Emulator Queueing System
- Dynamic Resource Management
- Result
- Conclusions
Backgrounds

- Trends – SoCs are in EVERYWHERE!

5G network  AI  IoT  Automotive

© Accellera Systems Initiative
Backgrounds

- Trends – SoCs must do EVERYTHING!

Modem, NPU / TPU, ISP / DSP, ADAS
Backgrounds

- Trends – Verification cost takes large portion of a pie!

SoC development cost breakdown

- **Verification**
  - Verification IP
  - Models
  - Emulation/FPGA flows
  - Arm approved design Services
  - Support & training

- **Software**
  - Debug tools
  - Compilers
  - Firmware
  - Apps
  - OS
  - Open source contributions
  - Models
  - FPGA/Emulation support
  - Software development boards

- **Physical design**
  - Reference methodologies
  - Physical IP and POP
  - DFT/DFM readiness and flows
  - Arm approved design services

- **Silicon IC prototyping**
  - Test methodologies
  - Models
  - Debug tools
  - Evaluation boards
  - Ready to use firmware or OS

- **IP qualification**
  - Silicon proven IP
  - Functional integration kits
  - Reference platforms
  - Models for simulation
  - Models for software acceleration
  - Support

- **Architecture/design**
  - Integration kits
  - Reference platforms
  - Models
  - Bus & Interface standards
  - Arm system IP
  - Functional safety support
  - Security – IP, software and platform
  - Emulation

Source: ARM
Backgrounds

- Problems
  - Different interfaces for different types of emulators
### Backgrounds

- **Problems**
  - Inefficient resource sharing

#### Starvation

- Average Pending Time (min)

#### Low Utilization

- Available resource and usage rate

© Accellera Systems Initiative
Emulator Queueing System

- Problems
  - Single interface for different types of emulators
Emulator Queueing System

- **Components**
  - Queue
  - Scheduler
  - Host Machines
  - Observer
  - User Interface
Emulator Queueing System

Resource Partitioning

- $N$ Partitions with $N$ type in an emulation farm
- Dedicated type for each partition

Type BIG

Type SMALL
Emulator Queueing System

- Resource Partitioning
  - Solution for starvation in warblers’ ecosystem
  - “Resource partitioning acts to promote the long-term coexistence of competing species.”
Emulator Queueing System

- Resource Partitioning
  - $N$ Partitions with $N$ types in an emulation farm
  - Dedicated type for each partition

Type BIG

Type SMALL
Emulator Queueing System

- Dynamic Resource Management
  - Dynamically configure the size of partitions

When type A and B requires similar resources

When type B requires more resources than Type A

When type A requires more resources than Type B
Emulator Queueing System

- Advantages
  - Easy to scale-out
Emulator Queueing System

- **Advantages**
  - Increase resource utilization
Emulator Queueing System

- Advantages
  - Increase resource utilization
Dynamic Resource Management

- What policy should be applied to make a decision?
Dynamic Resource Management

- What policy should be applied to make a decision?

1. Machine learning based policy
   - Reinforcement learning – Deep Q Network

2. Heuristic based policies
   - Quality of Service (QoS)
   - Greedy
   - Fair share
Dynamic Resource Management: Machine Learning Policy

- Why do we use reinforcement learning?
Dynamic Resource Management: Machine Learning Policy

- Why do we use reinforcement learning?

✓ Markov Decision Process

- Markov decision process (MDP) is a **discrete time** stochastic control process.
- Mathematical framework for **modeling decision making** in situations where outcomes are partly random and partly under the control of a decision maker.
- MDPs are useful for studying **optimization problems** solved via dynamic programming and **reinforcement learning**.

Dynamic Resource Management: Machine Learning Policy

- How does the reinforcement learning find out the optimal solution?
Dynamic Resource Management: Machine Learning Policy

- How does the reinforcement learning find out the optimal solution?
  - **Bellman Equation**

\[
V^\pi^*(s) = \max_a \{ R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^\pi^*(s') \}
\]

- \( V(s) \): value function
- \( s \): state
- \( s' \): next state
- \( a \): action
- \( R(s,a) \): reward function
- \( r \): discounted rate
- \( P(s'|s,a) \): conditional probability
Dynamic Resource Management: Machine Learning Policy

- How does the reinforcement learning find out the optimal solution?

✔ Bellman Equation

\[
V^\pi^*(s) = \max_a \{ R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^\pi^*(s') \}
\]

- V(s): value function
- s: state
- s': next state
- a: action
- R(s,a): reward function
- r: discounted rate
- P(s'|s,a): conditional probability
Dynamic Resource Management: Machine Learning Policy

- How does the reinforcement learning find out the optimal solution?
  - Bellman Equation

\[ V^\pi^*(s) = \max_a \{ R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^\pi^*(s') \} \]

- V(s): value function
- s: state
- s': next state
- a: action
- R(s,a): reward function
- r: discounted rate
- P(s'|s,a): conditional probability
How does the reinforcement learning find out the optimal solution?

✓ Bellman Equation

\[ V^{\pi*}(s) = \max_a \{ R(s, a) + \gamma \sum_{s'} P(s'|s, a)V^{\pi*}(s') \} \]

- \( V(s) \): value function
- \( s \): state
- \( s' \): next state
- \( a \): action
- \( R(s,a) \): reward function
- \( r \): discounted rate
- \( P(s|s,a) \): conditional probability
Dynamic Resource Management: Machine Learning Policy

- Application

Pending Info

Scores

Hidden Layers

Actions

argmax

Environment

Dynamic Resource Management: Machine Learning Policy
Dynamic Resource Management: Machine Learning Policy

- How did we improve learning speed?
  - Stay action when there is no pending jobs
  - Replay memory & Target network

* The concept is presented in the paper ‘Play Atari with deep reinforcement learning’ by Deepmind
Dynamic Resource Management: Machine Learning Policy

- Architecture

1. Store into replay memory for every step
2. Load random memory in batch size every T train steps for N iteration
3. Get loss from target network
4. Update neural network
5. Update target network for every S steps
6. Get action for max rewards
   Or get random action by E probability
Dynamic Resource Management: Machine Learning Policy

- **Architecture**

  1. Store into replay memory for every step
  2. Load random memory in batch size every $T$ train steps for $N$ iteration
  3. Get loss from target network
  4. Update neural network
  5. Update target network for every $S$ steps
  6. Get action for max rewards
     Or get random action by $E$ probability

---

© Accellera Systems Initiative
Dynamic Resource Management: Machine Learning Policy

- **Architecture**

1. **Emulator Queueing System**
   - Store into replay memory for every step

2. **Replay memory**
   - Batch Size
   - ...

3. **Main Network**
   - Get loss from target network
   - Update neural network

4. **Target Network**
   - Update target network for every S steps
   - Get action for max rewards
     - Or get random action by E probability

5. **Loss**

6. **Dynamic Resource Management:**
   - Machine Learning Policy
Dynamic Resource Management: Machine Learning Policy

- **Architecture**

  1. **Store into replay memory for every step**
  2. **Load random memory in batch size every T train steps for N iteration**
  3. **Get loss from target network**
  4. **Update neural network**
  5. **Update target network for every S steps**
  6. **Get action for max rewards or get random action by E probability**
Dynamic Resource Management: Machine Learning Policy

- Number of trains: 10,000
  - 5,000 episodes, 10 iterations

(A) Average reward while training

(B) Average reward for 500 tests
Dynamic Resource Management: Heuristic Policies

Quality of Service  Greedy  Fair Share
Dynamic Resource Management: Heuristic Policies

• **QoS – Max pending time**

**Algorithm 1** QoS policy

IF \( \max(\text{big-type job pending time}) > \max(\text{small-type job pending time}) \) THEN
  IF \( \max(\text{big-type job pending time}) > \text{BIG MAX} \) THEN
    allocate to big
  END IF
ELSE IF \( \max(\text{small-type job pending time}) > \text{SMALL MAX} \) THEN
  IF \( \max(\text{small-type job}) > \text{BIG MAX} \) THEN
    allocate to small
  END IF
END IF

*BIG MAX and SMALL MAX is a constant number.*

*Allocation scheme: Max-machine, max-unit*

**Decision interval: 5 min**
Dynamic Resource Management:
Heuristic Policies

• **Greedy → Current queue status**

Algorithm 2 Greedy policy

IF $\frac{\text{big}}{\text{total}}$ ratio > RATIO THEN
  IF $\frac{\text{big}}{\text{total}}$ pending time > BIG TOTAL MAX THEN
    allocate to big
  END IF
ELSE IF $\frac{\text{small}}{\text{total}}$ ratio > RATIO THEN
  IF $\frac{\text{small}}{\text{total}}$ time > SMALL TOTAL MAX THEN
    allocate to small
  END IF
END IF

*BIG RATIO, SMALL RATIO, BIG TOTAL MAX and SMALL TOTAL MAX are constant numbers.*

• **Fair Share → Average pending time**

Algorithm 3 Fair share policy

IF big-type jobs’ pending avg > small-type jobs’ pending avg THEN
  allocate to big
ELSE
  allocate to small
END IF
Dynamic Resource Management: Heuristic Policies

- Experimental Result

* Total pending time score = normalized value of total pending time

** Fairness score = normalized value of total pending time
Results

- Real Environment Result

*Algorithm*: QoS, Greedy and Fair share combination

(A) Max pending time

(B) Total pending time

(C) Average pending time ratio
Conclusions

The increase in emulation resource utilization indicates the increase in the number of jobs to run in emulation farms (20%)

*Estimated time: 10:00 ~ 20:00 (operation time)*
Conclusions

• Contribution
  1. Improve efficiency for emulator management system
  2. First definition for dynamic resource management policy on emulator management system
  3. First machine learning approach on emulation management system

• Future works
  1. Advanced Reinforcement learning (A3C …)
  2. Common computing farm with N partitions
Q & A

Or send an email to sangwoo.noh@samsung.com
Appendix : Comparison

Before – 2019.03.13 / After – 2019.07.02

<table>
<thead>
<tr>
<th>Number of jobs</th>
<th>big</th>
<th>small</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>before</td>
<td>46</td>
<td>148</td>
<td>194</td>
</tr>
<tr>
<td>after</td>
<td>53</td>
<td>146</td>
<td>199</td>
</tr>
<tr>
<td>diff (rate)</td>
<td>15.2%</td>
<td>-1.4%</td>
<td>2.6%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module size</th>
<th>big</th>
<th>small</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>before</td>
<td>406</td>
<td>456</td>
<td>862</td>
</tr>
<tr>
<td>after</td>
<td>520</td>
<td>540</td>
<td>1060</td>
</tr>
<tr>
<td>diff (rate)</td>
<td>28.1%</td>
<td>18.4%</td>
<td>23%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Average Run time(hours)</th>
<th>big</th>
<th>small</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>before</td>
<td>2</td>
<td>2.1</td>
<td>4.1</td>
</tr>
<tr>
<td>after</td>
<td>2</td>
<td>2.6</td>
<td>4.6</td>
</tr>
</tbody>
</table>

© Accellera Systems Initiative
Implementation