Pipelined Data-Parallel CPU/GPU Scheduling for Multi-DNN Real-Time Inference

Yecheng Xiang and Hyoseung Kim

UCRIVERSIDE

RTSS 2019
Introduction

- Running multiple DNN models on a heterogeneous platform (e.g., NVIDIA TX2: 4 Cortex-A57, 2 Denver CPU cores, and a Pascal GPU) is gaining much interest.
  - E.g., self-driving car: multiple sensing tasks, vision-based perception algorithms.
- To ensure the usefulness and correctness, timely DNN inference execution is a must.
  - A bounded tail latency, i.e., Worst-Case Response Time, is needed.
- Efficient utilization of computing resources is important.
Each DNN model is composed of several layers – input layer, hidden layers and output layer.

Execution pattern of an inference job: forward propagation from input to output layers.

Execution time and response time of each layer may differ.
The Status Quo

Modern deep learning inference frameworks (e.g., Torch, Tensorflow, and Caffe)

- Handle inference jobs in a sequential manner.

- One separate process per DNN model.
  - CPU: multi-threading BLAS libraries (e.g., OpenBLAS) offer limited control over real-time tasks accessing different DNN models.
  - GPU: concurrent GPU kernel execution due to different CUDA contexts.

- No prioritization or real-time support.
The Status Quo – CPU

(a) Status quo: multi-process version

(b) Status quo: single-process multithreading

Long Response Time

Long Blocking Time
The Status Quo – GPU

(a) Status quo: single CUDA stream (single-process)

(b) Status quo: different CUDA contexts (multi-process)

No Concurrent Kernel Execution

Long Response Time
Prior Work

- **S³DNN¹**, Case Study in autonomous driving applications²
  - Efforts towards improving the average-case response time by improving scheduling on GPUs (e.g., supervised streaming, pipelining, and parallelism).
  - Only considers one type of DNN model, no concurrent requests to multiple DNN models.
  - Only GPU is considered and utilized.

- **Glimpse³, MCDNN⁴**
  - Mobile DNN frameworks collaborating with the cloud to improve latency.
  - No real-time schedulability guarantee.

- **DNN Optimization Techniques⁵,⁶**
  - Compressing DNN models or layers, trading off output accuracy for performance gain.
  - Reduces the execution time of individual jobs, but the scheduling problem of concurrent jobs remains.

---

Our Contributions

**DART: a real-time multi-DNN inference framework for heterogeneous CPU/GPU platforms**

- Brings algorithmic improvements into real-time DNN scheduling.
- First work to bound WCRT and ensure schedulability with high throughput.

**Details:**

- Introduces new abstractions to deal with different resource requirements of layers of DNNs and to facilitate the co-utilization of CPU and GPU.
- Gives a systematic formulation of real-time DNN scheduling as a distributed acyclic scheduling problem.
- Develops resource management algorithms to i) balance the contention across processors and ii) allocate resources to ensure and improve real-time schedulability and response time.
System Model

- Heterogeneous multi-core system with single GPU
- Main memory is shared among all CPUs
- Task type: Real-time (RT), best-effort (BE)

General Task Model

$$\tau_i := (C_i, T_i, D_i, L_i)$$

WCET, min inter-arrival time, deadline, #of layers

GPU Task Model

$$C_{i,j}(p_k) := (G_{i,j}^{hd}(p_k), G_{i,j}^e(p_k), G_{i,j}^m(p_k), G_{i,j}^{dh}(p_k))$$

H2D memcpy, Kernel execution, misc. CPU operations, D2H memcpy
### Key Ideas:

- Partition layers into multiple **stages**.

- A stage can be processed on different computing **nodes**.

- Dynamic-programming based approach to match **stages to nodes** to maximize the performance while meeting the real-time constraints.
- Partition the DNN models of each task $\tau_i$ into stages
Assign the stages of different tasks to the nodes. Note that stages can skip nodes when needed. (The assignment is determined by the allocation algorithm.)
Scheduling Architecture III

- Two task classes – RT and BE, where RT is strictly prioritized over BE.

- RT workers and BE workers for each task class.
  - Each RT/BE worker is statically allocated to its allocated node for the execution of RT/BE tasks respectively.
  - RT workers can preempt BE workers on the same node.
  - Task execution within each worker is non-preemptive.

- RT worker uses **deadline-monotonic (DM)** scheduling policy.
  - Deterministic guarantees under overload condition
- BE worker uses **earliest-deadline-first (EDF)** scheduling policy.
  - Higher utilization

- **Batched execution** is also enabled for BE tasks.
  - Maximize throughput
CPU Scheduling Example

Two type of Tasks: RT & BE
Task preemption

Developed OpenBLAS-rt: a real-time extension of OpenBLAS

RT task deadline is met
Better CPU utilization
GPU Scheduling Example

CUDA streams + shared CUDA context
-> Concurrent kernel execution

High-priority CUDA stream
-> Improved response time

Multiple low-priority CUDA streams ->
Improved throughput and GPU utilization
Allocation Algorithm -- Designing Task Stages

**Goals:**
- Construct the stages of task $\tau_i$
- Allocate each stage to a node
- Balance utilization of nodes after allocation
  - *Reduce contention on nodes.*

**Approach:** Formulate as a dynamic programming algorithm

- $M[n, k]$ denotes the utilization of the most loaded node when the first $n$ layers of a task $\tau_i$ are allocated to the first $k$ nodes.

\[
M[n, k] = \min_{x=0}^{n} \max(M[x, k-1], w[k] + \sum_{y=x+1}^{n} U_{i,y}(p_k)) \tag{7}
\]

\[
M[0, k] = 0,
\]

\[
M[1, k] = \min_{q=1}^{k}(w[q] + U_{i,1}(p_q))
\]

Solution and stage allocation are found by: $M[n = L_i, k = N_p]$. 

16
Allocation Algorithm -- Finding a Node Configuration

Algorithm 1 Find a Node Configuration for Tasks

Require: \( \Gamma = \{ \tau_1, \tau_2, \tau_3, \ldots, \tau_n \} \): taskset
Require: \( \mathbb{P} \): a set of candidate node configurations
Ensure: \( P_{\text{sol}} \): a node config. found (solution); \( P_{\text{sol}} = \emptyset \), if failed.

1: function FIND_NODE_CONFIGURATION(\( \Gamma \), \( \mathbb{P} \))
2: \( P_{\text{sol}} = \emptyset /* initialization */ 
3: \( W_{\text{sol}} = \infty /* weighted response time of RT tasks for \( P_{\text{sol}} \) */ 
4: for all \( P \in \mathbb{P} \) do
5: \( N_P = |P| 
6: \text{Initialize } w[1...N_P] 
7: for all \( \tau_i \in \Gamma \) in descending order of \( U_i^{\text{avg}} \) do
8: \( L_i = \) the number of layers of \( \tau_i \)
9: \( M[L_i, N_P] \) for \( \tau_i \) by Eq. (7)
10: Store the stage-to-node allocation of \( \tau_i \)
11: Update \( w[1...N_P] \) with \( \tau_i \)
12: \( \Gamma^{\text{RT}} = \) a set of all RT tasks in \( \Gamma \)
13: if \( \forall \tau_i \in \Gamma^{\text{RT}} \) passes the schedulability test of Eq. (5) then
14: \( \forall \tau_i \in \Gamma^{\text{RT}}, R_i = \) worse-case response time of \( \tau_i \)
15: \( W = \sum_{\tau_i \in \Gamma^{\text{RT}}} (\pi_i/|\Gamma^{\text{RT}}|) \times (R_i/D_i) /* \pi_i: priority */ 
16: if \( W < W_{\text{sol}} \) then
17: \( P_{\text{sol}} = P 
18: W_{\text{sol}} = W 
19: end function

Task average utilization:
\[
U_i^{\text{avg}} = (1/k) \cdot \sum U_i(p_k)
\]

Weighted WCRT:
\[
W = \sum_{\tau_i \in \Gamma^{\text{RT}}} (\pi_i/|\Gamma^{\text{RT}}|) \times (R_i/D_i) /* \pi_i: priority */
\]
Schedulability Analysis

- The execution sequence of stages of a task across nodes can be modeled as a directed acyclic path in a graph (DAG) of nodes.
- We use the schedulability analysis in [1] which is based on the non-preemptive DAG delay composition theorem and reduces the DAG into an equivalent uniprocessor system.
- We apply analysis to our system and bound the worst-case response time of $R_i$ of a task $\tau_i$ by:

$$R_i^{(0)} = C_e^*(i); \quad R_i^{(k)} = C_e^*(i) + \sum_{\tau_h \in h_p(\tau_i)} \left[ \frac{R_i^{(k-1)}}{T_h} \right] C_h^*$$

DART – Miscellaneous Components

i) Layer-wise Execution Time Profiling
   - Construct a WCET database for DART runtime.
   - Estimate WCET of individual layers for candidate node configurations.

ii) Admission Control
   - Ensure the schedulability of all the admitted RT tasks.

iii) Run-time Task Enforcement
   - Run-time execution time of each task monitored by DART.
   - DART demotes a RT task to BE task if it detects a execution time exceedance.
   - For execution time exceedance of BE tasks, DART changes deadline to infinite to mitigate the impact on other BE tasks.
Evaluation – Baselines

- BaseCPU
- BaseGPU
  - They both **represent state-of-the-art** inference frameworks. (e.g., TensorRT Inference Server and TensorFlow Serving)
  - **One run queue per DNN model**
  - **One process/instance per DNN model**
  - **Priority queue added** to make a fair comparison with DART (RT tasks are prioritized over BEs while waiting in the queue)
Evaluation -- Experiment Setup

### Hardware

- X86 Server
  - Xeon 8-core 2.1GHZ E2620 v4 CPU
  - 32GB RAM
  - GTX 1080

- ARM Server (NVIDIA TX2)
  - 4 ARM Cores (Quad ARM® A57)
  - 2 HMP Denver 2 Cores
  - Integrated Pascal GPU

### DNN Models

- Alexnet
- Lenet
- VGGnet
- PilotNet
Evaluation – Runtime Overhead

TABLE I: Inter-node communication overhead on TX2

<table>
<thead>
<tr>
<th>Time (us)</th>
<th>A57-A57</th>
<th>A57-Den</th>
<th>Den-A57</th>
<th>Den-Den</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>20.83</td>
<td>36.33</td>
<td>42.58</td>
<td>51.45</td>
</tr>
<tr>
<td>Maximum</td>
<td>48</td>
<td>69</td>
<td>82</td>
<td>102</td>
</tr>
</tbody>
</table>

TABLE II: CUDA stream kernel preemption overhead

<table>
<thead>
<tr>
<th>Time (us)</th>
<th>GTX 1080</th>
<th>TX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>17.87</td>
<td>28.76</td>
</tr>
<tr>
<td>Maximum</td>
<td>52</td>
<td>121</td>
</tr>
</tbody>
</table>

- Communication overhead is highest between two Denver cores, however it is still acceptably small compared to typical DNN execution time.
- Preemption overhead is marginable, yet is still modeled in $\epsilon^{gp}$ in Eq. (4).
• Overall, the **speedup diminishes** while the number of cores increases.
• Speedup from an increased number of CPU cores varies significantly by layer.
  • E.g., layer 3 & 5 do not get noticeable benefit by increasing the number of A57 cores. While layer 2 and 4 have speedups with more CPUS.
• Performance differs significantly among different types of processors.
  • GPU has the best performance for most layers.
  • Denver CPU cores overall perform better than ARM A57 CPUs.
  • However, GPUs might be slower than CPUs for some layers. (Layer 5)
Evaluation – Schedulability Experiments

- Results are based on random-generated tasksets.
- DART dominates the baseline in schedulability.
Evaluation – Response Time and Throughput

- Consider a mixture of real-time (RT) and best-effort (BE) tasks on Xeon and TX2 platforms.
- We plot the response time CDF of the RT tasks and measure the throughput of BE tasks.
- We enable batching when executing on GPU with configurable batch sizes of 1, 8, 16, 32.

**TABLE III: Taskset information on Xeon and TX2**

<table>
<thead>
<tr>
<th>Task name</th>
<th>DNN model</th>
<th>Deadline</th>
<th>Class</th>
<th>RT Prio</th>
</tr>
</thead>
<tbody>
<tr>
<td>pilot_rt_1</td>
<td>Pilotnet</td>
<td>150 ms</td>
<td>RT</td>
<td>90</td>
</tr>
<tr>
<td>pilot_rt_2</td>
<td>Pilotnet</td>
<td>150 ms</td>
<td>RT</td>
<td>89</td>
</tr>
<tr>
<td>alexnet_rt_1</td>
<td>Alexnet</td>
<td>200 ms</td>
<td>RT</td>
<td>88</td>
</tr>
<tr>
<td>alexnet_rt_2</td>
<td>Alexnet</td>
<td>200 ms</td>
<td>RT</td>
<td>87</td>
</tr>
<tr>
<td>pilot_be_1</td>
<td>Pilotnet</td>
<td>–</td>
<td>BE</td>
<td>-</td>
</tr>
<tr>
<td>alexnet_be_1</td>
<td>Alexnet</td>
<td>–</td>
<td>BE</td>
<td>-</td>
</tr>
<tr>
<td>lenet_be_1</td>
<td>LeNet</td>
<td>–</td>
<td>BE</td>
<td>-</td>
</tr>
</tbody>
</table>

The jobs of the BE tasks arrive in a back-to-back manner to maximize the throughput.
DART achieves **98.5%** reduction in the maximum observed response time from BaseGPU with the batch size of 32.

(b) CDF of alexnet_rt_2
Evaluation – Response Time and Throughput

- The throughput improvement from batching diminishes after batch size reaches 16.
- DART achieves as much as 17.9% higher throughput than BaseGPU for `alexnet_be_1` with the batch size of 32.

Note: In the figure, the $b_x$ denotes the execution is with a batch size of $x$.

More results can be found in the paper.
Conclusions

- We present DART, a real-time DNN inference framework that offers deterministic response time and schedulability analysis to real-time DNN inference tasks and supports scheduling concurrent execution of various DNN models.

- We have implemented the scheduling architecture of DART and its key components, including pipeline stage design, node configuration, execution time profiling, admission control, and runtime enforcement, on Intel Xeon and Nvidia TX2 platforms.

- Experimental results have shown that DART dominates the baselines in both real-time performance and throughput.
Future Work

- Remote machine can be modeled as one node in the pipeline thus can be used to address the computational limit of local hardware.

- Other hardware accelerators can be utilized (e.g., FPGAs).

- Shared memory resources (e.g., caches and memory buses) and their performance interference are worth investigating to improve the CPU parallelization performance.
THANK YOU!

Questions?

Email us at: yxian013@ucr.edu, hyoseung@ucr.edu