OpenVX and Real-Time Certification: The Troublesome History

Tanya Amert, Sergey Voronov, and James H. Anderson

University of North Carolina at Chapel Hill
Motivation

Many of these processes can be represented as graphs.

Can we guarantee response times bounds for them all?
The Challenge

Tracking requires predicting the next position from a series of past positions. This results in a cycle in the graph.

Rabbit Tracking

- Regular edge
- Delay edge

Image ➔ Detect rabbits ➔ Match detections & predictions ➔ Predict new locations ➔ Update tracks ➔ Trajectories

A delay edge indicates results from prior time steps are needed.
The Challenge

Tracking requires predicting the next position from a series of past positions. This results in a cycle in the graph.

Existing schedulability analysis precludes cycles:
- requires data to be very old, or
- requires cycle utilization to be at most 1.0.

GPU sharing can result in cycles with utilization exceeding 1.0.
The Challenge

Tracking requires predicting the next position from a series of past positions. This results in a cycle in the graph.

Existing schedulability analysis precludes cycles:
- requires data to be very old, or
- requires cycle utilization to be at most 1.0.

GPU sharing can result in cycles with utilization exceeding 1.0.

We need response-time analysis that allows for arbitrary cycles in graph-based task systems.
Outline

• Motivation

• Transforming an OpenVX graph to independent sporadic tasks

• Bounding response-times for rp-sporadic tasks

• Evaluation

• Future work
Transformation from graph to tasks

Prior work has shown how to transform from an OpenVX graph to a set of independent sporadic tasks.

Before describing how we modify this transformation, we will overview the existing steps.

M. Yang et al.
RTSS ‘18
Starting point: OpenVX

OpenVX is a standard for specifying computer vision applications

– graph-based: primitives act on data objects (regular or delay)
– delay objects represent historical dependencies
Step 1: Coarse- to Fine-Grained OpenVX

OpenVX primitives can be too coarsely grained
- idea: break primitives into CPU and GPU nodes

[M. Yang et al., RTSS ’18]
Step 1: Coarse- to Fine-Grained OpenVX

OpenVX primitives can be too coarsely grained
- idea: break primitives into CPU and GPU nodes

[M. Yang et al., RTSS '18]
Transformation from graph to tasks

Prior work has shown how to transform from an OpenVX graph to a set of independent sporadic tasks.

Motivation

Transformation

Bounding response times

Evaluation

Future Work

Coarse-Grained OpenVX Graph $G^i$

Step 1

Fine-Grained OpenVX Graph $G^i$

Step 2

Sporadic Task Graph $\Gamma^i$

Step 3

Sporadic Task DAG $\tau^i$

Step 4

Sporadic Task Set $\tau^i$

M. Yang et al.
RTSS ‘18

K. Yang et al.
RTNS ‘15
Step 2: Fine-Grained OpenVX graph $\mathcal{G}$ to sporadic task graph $\Gamma$

- One-to-one mapping of nodes

![Graph diagram](image-url)
Step 2: Fine-Grained OpenVX graph $\mathcal{G}$ to sporadic task graph $\Gamma$

- One-to-one mapping of nodes

[K. Yang et al., RTNS ’15]
Step 2: Fine-Grained OpenVX graph $G$ to sporadic task graph $\Gamma$

- One-to-one mapping of nodes
- Data-to-node edges in $G$ become edges in $\Gamma$

[K. Yang et al., RTNS ’15]
Step 2: Fine-Grained OpenVX graph $\mathcal{G}$ to sporadic task graph $\Gamma$

- One-to-one mapping of nodes
- Data-to-node edges in $\mathcal{G}$ become edges in $\Gamma$

[K. Yang et al., RTNS ’15]
Step 2: Fine-Grained OpenVX graph $G$ to sporadic task graph $\Gamma$

- One-to-one mapping of nodes
- Data-to-node edges in $G$ become edges in $\Gamma$

[K. Yang et al., RTNS ’15]
Step 2: Fine-Grained OpenVX graph $\mathcal{G}$ to sporadic task graph $\Gamma$

- One-to-one mapping of nodes
- Data-to-node edges in $\mathcal{G}$ become edges in $\Gamma$
  - edges of the same type can be combined

Transformation

Bounding response times

Evaluation

Future Work

[K. Yang et al., RTNS '15]
Step 2: Fine-Grained OpenVX graph $\mathcal{G}$ to sporadic task graph $\Gamma$

- One-to-one mapping of nodes
- Data-to-node edges in $\mathcal{G}$ become edges in $\Gamma$
  - edges of the same type can be combined

\[ p = q = 1 \quad \text{and} \quad p = 2, q = 3 \]

[K. Yang et al., RTNS '15]
Step 2: Fine-Grained OpenVX graph $\mathcal{G}$ to sporadic task graph $\Gamma$

- One-to-one mapping of nodes
- Data-to-node edges in $\mathcal{G}$ become edges in $\Gamma$
  - edges of the same type can be combined

$\tau_1 \rightarrow \tau_3 \rightarrow \tau_2 \rightarrow \tau_4 \rightarrow \tau_5 \rightarrow \tau_6$

$p = 2, q = 3$

$p = q = 1$

[K. Yang et al., RTNS '15]
Transformation from graph to tasks

Prior work has shown how to transform from an OpenVX graph to a set of independent sporadic tasks.

Coarse-Grained OpenVX Graph $G^i$ \rightarrow Fine-Grained OpenVX Graph $G^i$ \rightarrow Sporadic Task Graph $\Gamma^i$ \rightarrow Sporadic Task DAG $\tau^i$ \rightarrow Sporadic Task Set $\tau^i$

M. Yang et al. RTSS ‘18 \rightarrow K. Yang et al. RTNS ‘15 \rightarrow K. Yang et al. RTNS ‘15
Step 3: Sporadic task graph $\Gamma$ to sporadic task \( \tau \)

- Forward delay edges can be dropped

\[\begin{align*}
&\tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \\
&\tau_3 \rightarrow \tau_4 \rightarrow \tau_5 \\
&\tau_5 \rightarrow \tau_6
\end{align*}\]

\[p = q = 1, \quad p = 2, q = 3\]

[K. Yang et al., RTNS '15]
Step 3: Sporadic task graph $\Gamma$ to sporadic task DAG $\tau$

- Forward delay edges can be dropped

$\tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \rightarrow \tau_4 \rightarrow \tau_5 \rightarrow \tau_6$

$p = 2, q = 3$

[K. Yang et al., RTNS '15]
Step 3: Sporadic task graph $\Gamma$ to sporadic task DAG $\tau$

- Forward delay edges can be dropped
- Cycles (due to backward delay edges) are combined into supernodes

[K. Yang et al., RTNS '15]
Step 3: Sporadic task graph $\Gamma$ to sporadic task DAG $\tau$

- Forward delay edges can be dropped
- Cycles (due to backward delay edges) are combined into supernodes

[K. Yang et al., RTNS ’15]
Transformation from graph to tasks

Prior work has shown how to transform from an OpenVX graph to a set of independent sporadic tasks.

Coarse-Grained OpenVX Graph $G_i$  Step 1  Fine-Grained OpenVX Graph $G_i$  Step 2  Sporadic Task Graph $\Gamma^i$  Step 3  Sporadic Task DAG $\tau^i$  Step 4  Sporadic Task Set $\tau^i$

M. Yang et al.  RTSS ‘18  K. Yang et al.  RTNS ‘15  K. Yang et al.  RTNS ‘15  Liu and Anderson  RTCSA ‘11
Step 4: Sporadic task DAG $\tau$ to independent sporadic tasks

[Amert, Voronov, and Anderson, RTCSA 2011]

![Diagram of task dependencies]

$Liu$ and Anderson, RTCSA 2011

Transformation
Bounding response times
Evaluation
Future Work

Motivation

CPU execution
GPU execution

Job Release
Job Deadline
Job Completion

Time

0 5 10 15 20 25 30

$\tau_1$  $\tau_2$  $\tau_3$  $\tau_456$  $\tau_456$
Step 4: Sporadic task DAG $\tau$ to independent sporadic tasks

[Amert, Voronov, and Anderson, RTCSA 2011]

$$\Phi_1 = 0$$

- $\tau_1$
- $\tau_2$
- $\tau_3$
- $\tau_4$ $\tau_5$ $\tau_6$

CPU execution: Red

GPU execution: Yellow

$\tau_1$ to independent sporadic tasks

$\tau_1$

Job Release

Job Deadline

Job Completion

Time

Motivation

Transformation

Bounding response times

Evaluation

Future Work
Step 4: Sporadic task DAG $\tau$ to independent sporadic tasks

[Liu and Anderson, RTCSA 2011]
Step 4: Sporadic task DAG $\tau$ to independent sporadic tasks

[Liu and Anderson, RTCSA 2011]
Step 4: Sporadic task DAG $\tau$ to independent sporadic tasks

[Liu and Anderson, RTCSA 2011]
Step 4: Sporadic task DAG $\tau$ to independent sporadic tasks

[Liu and Anderson, RTCSA 2011]
Step 4: Sporadic task DAG $\tau$ to independent sporadic tasks

[Liu and Anderson, RTCSA 2011]
Step 4: Sporadic task DAG $\tau$ to independent sporadic tasks

$Liu$ and Anderson, RTCSA 2011

CPU execution

GPU execution

graph end-to-end response time bound: 25
Step 4: Sporadic task DAG $\tau$ to independent sporadic tasks

[Liu and Anderson, RTCSA 2011]
Transformation from graph to tasks

Prior work has shown how to transform from an OpenVX graph to a set of independent sporadic tasks.

Coarse-Grained OpenVX Graph $G^i$ → Fine-Grained OpenVX Graph $G^i$ → Sporadic Task Graph $\Gamma^i$ → Sporadic Task DAG $\tau^i$ → Sporadic Task Set $\tau^i$

Step 1: M. Yang et al. RTSS ‘18
Step 2: K. Yang et al. RTNS ‘15
Step 3: K. Yang et al. RTNS ‘15
Step 4: Liu and Anderson RTCSA ‘11

Choice of intra-task parallelism
Choices of intra-task parallelism

Let $P_i$ be the intra-task parallelism allowed for task $\tau_i$.

Sequential sporadic tasks assume no intra-task parallelism:

$$P_i = 1.$$
Choices of intra-task parallelism

Let $P_i$ be the intra-task parallelism allowed for task $\tau_i$.

Sequential sporadic tasks assume no intra-task parallelism:

$P_i = 1$.

Parallel sporadic tasks assume full intra-task parallelism:

$P_i = \#CPUs = m$. 
The problem

Let $P_i$ be the intra-task parallelism allowed for task $\tau_i$.

**Sequential:** if the cycle utilization is greater than 1.0, response times can be unbounded.

**Full parallelism:** precedence constraints (p of delay edge) may not be respected.
Contribution: restricted parallelism

Let $P_i$ be the intra-task parallelism allowed for task $\tau_i$.

**Restricted parallelism:** allow supernodes to execute with restricted intra-task parallelism.

$$P_i = p$$
Contribution: restricted parallelism

Let $P_i$ be the intra-task parallelism allowed for task $\tau_i$.

Restricted parallelism: allow supernodes to execute with restricted intra-task parallelism.
- Respects precedence constraints 😊
- Better response times than sequential 😊
- Impacts accuracy 😞
Outline

• Motivation
• Transforming an OpenVX graph to independent sporadic tasks
  • **Bounding response-times for rp-sporadic tasks**
• Evaluation
• Future work
The rp-sporadic task model

We add an additional parameter to the sporadic task model:

\[ \tau_i = (\Phi_i, T_i, C_i, P_i) \]

\( P_i \) corresponds to the \( p \) value from OpenVX.

Note: consider all nodes as CPU.

(we’ll come back to this)
The rp-sporadic task model

We add an additional parameter to the sporadic task model:

\[ \tau_i = (\Phi_i, T_i, C_i, P_i) \]

\( P_i \) corresponds to the \( p \) value from OpenVX.

Feasibility conditions:

- \( \sum_i u_i \leq m \)
  - same as in either sequential or fully parallel task models
- \( \forall i : u_i \leq P_i \)
  - generalizes to sequential execution if \( p = 1 \), or full parallelism if \( p = m \)

Note: consider all nodes as CPU.

(we’ll come back to this)
Bounding response times

Definitions:

Let $J$ be the job of interest.

Assume EDF scheduling:

- Let $HP$ be the set of jobs with deadlines at or before $J$
- Let $LP$ be the set of jobs with deadlines after $J$

Assumptions:

- GPU use is arbitrated by a mutex lock
  - GPU execution time is included in CPU worst-case execution time
- A task $\tau_i$ can have up to $P_i$ jobs scheduled concurrently
- A job in $LP$ can have a non-preemptive section (e.g., GPU use)
  - We’ll call these $np$-sections
Bounding response times – proof overview

- $t_0$: Release of $J$
- Deadline of $J$
- Completion of $J$

- busy interval
- non-busy interval
Bounding response times – proof overview

- No new jobs with priority higher than $J$ are released.
Bounding response times – proof overview

- No new jobs with priority higher than $J$ are released
- Inductively apply a response-time bound $X$ for higher-priority jobs

Timeline:
- $t_0$: Release of $J$
- Deadline of $J$
- Completion of $J$

Busy interval: $t_0$ to Deadline of $J$
Non-busy interval: Deadline of $J$ to Completion of $J$
Bounding response times – proof overview

- No new jobs with priority higher than $J$ are released
- Inductively apply a response-time bound $X$ for higher-priority jobs
- Determine how far these jobs push back $J$

$t_0$: Release of $J$
Deadline of $J$
Completion of $J$

Busy interval
Non-busy interval
Bounding response times – proof overview

- Jobs ready for scheduling
- Scheduled np-sections of LP jobs
- Jobs whose predecessors are not completed

No new jobs with priority higher than $J$ are released

Inductively apply a response-time bound $X$ for higher-priority jobs

Determine how far these jobs push back $J$

- Release of $J$
- Deadline of $J$
- Completion of $J$

$\tau_0$

Busy interval

Non-busy interval

Motivation
Transformation
Bounding response times
Evaluation
Future Work
Bounding response times – proof overview

- **Non-completed workload**: $P_i - 1$ jobs for task $\tau_i$
- **Jobs ready for scheduling**
- **Scheduled np-sections of LP jobs**
- **Jobs whose predecessors are not completed**

- **At most one np-section per ready job**
- **At most $m - 1$ tasks**

- **Release of $J$**
- **Deadline of $J$**
- **Completion of $J$**

- **Inductively apply a response-time bound $X$ for higher-priority jobs**
- **Determine how far these jobs push back $J$**

- **Transformation**
- **Bounding response times**
- **Evaluation**
- **Future Work**
Bounding response times – proof overview

- Jobs whose predecessors are not completed
- Scheduled np-sections of LP jobs
- At most one np-section per ready job
- Release of $J$
- Deadline of $J$
- Completion of $J$
- Time

**Motivation**

- Inductively apply a response-time bound $X$ for higher-priority jobs
- Determine how far these jobs push back $J$

**Future Work**

- Bounding response times
- Evaluation
- OpenVX and Real-Time Certification: The Troublesome History

Amert, Voronov, and Anderson

University of North Carolina at Chapel Hill
Bounding response times – example schedule

- Scheduled $HP$ jobs
- Scheduled np-sections of $LP$ jobs
- $J$

Motivation
Transformation
Bounding response times
Evaluation
Future Work
Bounding response times – example schedule

- Scheduled HP jobs
- Scheduled np-sections of LP jobs
- $J$

Motivation
Transformation
Bounding response times
Evaluation
Future Work

Schedule

$\begin{align*}
\text{Release of } J \\
\text{Deadline of } J \\
\text{Completion of } J \\
\end{align*}$

Busy interval
Non-busy interval

$m$ processors
Bounding response times – example schedule

- Scheduled $HP$ jobs
- Scheduled np-sections of $LP$ jobs
- $J$

Motivation
Transformation
Bounding response times
Evaluation
Future Work

$\text{Release of } J$
$\text{Deadline of } J$
$\text{Completion of } J$

busy interval
non-busy interval

Time

$0$

$m$ processors

single np-section of a job with deadline after $J$
Bounding response times – example schedule

Scheduled *HP* jobs

Scheduled np-sections of *LP* jobs

*J*

potential interference due to an np-section of an *LP* job, which has to be scheduled at *t₀*

no interference due to jobs of other tasks

single np-section of a job with deadline after *J*

Transformation

Bounding response times

Evaluation

Future Work

*Motivation*

Transformation

*Bounding response times*

Evaluation

Future Work

**Motivation**

Transformation

**Bounding response times**

Evaluation

Future Work

Transformation

**Bounding response times**

Evaluation

Future Work
Bounding response times – example schedule

Computing a response-time bound:
1. Bound total interfering workload at $t_0$
2. Compute interfering workload at the release of $J$: $L(X)$
3. Obtain inductive condition ($J$ finishes within $X$ time units after release):
   $$L(X) \leq mX$$
Outline

• Motivation
• Transforming an OpenVX graph to independent sporadic tasks
• Bounding response-times for rp-sporadic tasks
• Evaluation
• Future work
Response-times versus history

- Case study: pedestrian tracking
  - Competing workloads can add excessive blocking on the GPU, resulting in cycle utilization greater than 1.0

<table>
<thead>
<tr>
<th></th>
<th>Sequential</th>
<th>( p = 1 )</th>
<th>( p = 2 )</th>
<th>( p = 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analytical Bound (ms)</td>
<td>N/A</td>
<td>N/A</td>
<td>927.27</td>
<td>928.37</td>
</tr>
<tr>
<td>Observed Max. Response Time (ms)</td>
<td>25250.67</td>
<td>572.81</td>
<td>713.53</td>
<td>537.60</td>
</tr>
<tr>
<td>Observed Avg. Response Time (ms)</td>
<td>11765.23</td>
<td>293.63</td>
<td>280.86</td>
<td>293.07</td>
</tr>
</tbody>
</table>
Response-times versus history

• Case study: pedestrian tracking
  – Competing workloads can add excessive blocking on the GPU, resulting in cycle utilization greater than 1.0

<table>
<thead>
<tr>
<th></th>
<th>Sequential</th>
<th>$p = 1$</th>
<th>$p = 2$</th>
<th>$p = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analytical Bound (ms)</td>
<td>N/A</td>
<td>N/A</td>
<td>927.27</td>
<td>928.37</td>
</tr>
<tr>
<td>Observed Max. Response Time (ms)</td>
<td>25250.67</td>
<td>572.81</td>
<td>713.53</td>
<td>537.60</td>
</tr>
<tr>
<td>Observed Avg. Response Time (ms)</td>
<td>11765.23</td>
<td>293.63</td>
<td>280.86</td>
<td>293.07</td>
</tr>
</tbody>
</table>

Observation: The system is unschedulable if the supernode is not replicated ($p = 1$) or if the graph is scheduled sequentially.
Response-times versus history

- Case study: pedestrian tracking
  - Competing workloads can add excessive blocking on the GPU, resulting in cycle utilization greater than 1.0

<table>
<thead>
<tr>
<th></th>
<th>Sequential</th>
<th>$p = 1$</th>
<th>$p = 2$</th>
<th>$p = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analytical Bound (ms)</td>
<td>N/A</td>
<td>N/A</td>
<td>927.27</td>
<td>928.37</td>
</tr>
<tr>
<td>Observed Max. Response Time (ms)</td>
<td>25250.67</td>
<td>572.81</td>
<td>713.53</td>
<td>537.60</td>
</tr>
<tr>
<td>Observed Avg. Response Time (ms)</td>
<td>11765.23</td>
<td>293.63</td>
<td>280.86</td>
<td>293.07</td>
</tr>
</tbody>
</table>

Observation: The analytical response-time bounds upper-bounded the observed response times for $p \geq 2$. 
Response-times versus history

- Case study: pedestrian tracking
  - Competing workloads can add excessive blocking on the GPU, resulting in cycle utilization greater than 1.0

<table>
<thead>
<tr>
<th></th>
<th>Sequential</th>
<th>( p = 1 )</th>
<th>( p = 2 )</th>
<th>( p = 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analytical Bound (ms)</td>
<td>N/A</td>
<td>N/A</td>
<td>927.27</td>
<td>928.37</td>
</tr>
<tr>
<td>Observed Max. Response Time (ms)</td>
<td>25250.67</td>
<td>572.81</td>
<td>713.53</td>
<td>537.60</td>
</tr>
<tr>
<td>Observed Avg. Response Time (ms)</td>
<td>11765.23</td>
<td>293.63</td>
<td>280.86</td>
<td>293.07</td>
</tr>
</tbody>
</table>

Observation: For \( p \geq 2 \), as \( p \) increases, the observed maximum response times decrease.
Response-times versus history

• Case study: pedestrian tracking
  – Competing workloads can add excessive blocking on the GPU, resulting in cycle utilization greater than 1.0

<table>
<thead>
<tr>
<th></th>
<th>Sequential</th>
<th>$p = 1$</th>
<th>$p = 2$</th>
<th>$p = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analytical Bound (ms)</td>
<td>N/A</td>
<td>N/A</td>
<td>927.27</td>
<td>928.37</td>
</tr>
<tr>
<td>Observed Max. Response Time (ms)</td>
<td>25250.67</td>
<td>572.81</td>
<td>713.53</td>
<td>537.60</td>
</tr>
<tr>
<td>Observed Avg. Response Time (ms)</td>
<td>11765.23</td>
<td>293.63</td>
<td>280.86</td>
<td>293.07</td>
</tr>
</tbody>
</table>

**Observation:** The analytical response-time bounds are almost identical for $p \geq 2$, as the number of tasks with $P_i < m$ is unchanged.
Accuracy versus history

- We expect the accuracy to decrease as $p$ increases
  - The distance between the “current” position of a pedestrian and their last-tracked position increases

- In our experiments, $p$ represents the actual age of history, rather than the maximum age
  - E.g., for $p = 2$, data produced by frames 0,2,4,6,... is never available to frames 1,3,5,7,...
Accuracy versus history: $p = 2$

- Accuracy metric: total number of tracks throughout the video
  - Success: track count is similar to that of $p = 1$

Observation: Accuracy is comparable for $p = 1$ and $p = 2$. Both sets of frames result in similar track counts.
Accuracy versus history: $p = 3$

- Accuracy metric: total number of tracks throughout the video
  - Success: track count is similar to that of $p = 1$

![Graph showing accuracy metric for different values of $p$]

Observation: Accuracy significantly decreases for $p = 3$.
Outline

• Motivation
• Transforming an OpenVX graph to independent sporadic tasks
• Bounding response-times for rp-sporadic tasks
• Evaluation
• Future work
Future Work

- **Remove assumptions** in analysis
- Extend experiments to consider **higher-level notions of accuracy** (e.g., number of obstacles not detected)
- Perform a **large-scale study of the trade-off** between response times, history, and accuracy
- Develop a tool to enable computer-vision programmers to graphically specify OpenVX programs, which are then automatically transformed to fine-grained implementations with response-time analysis
Summary

• Introduced rp-sporadic task model
  – Provided response-time bound derivation
  – First work to bound response-times for graphs with arbitrary cycles
• Preliminary evaluation of history/accuracy/response-time trade-off

Questions? Please let us know!
{amert,rdkl,anderson}@cs.unc.edu