Making OpenVX Really “Real Time”

Ming Yang¹, Tanya Amert¹, Kecheng Yang¹,², Nathan Otterness¹, James H. Anderson¹, F. Donelson Smith¹, and Shige Wang³

¹The University of North Carolina at Chapel Hill
²Texas State University
³General Motors Research
700 ms
A new approach for OpenVX™ graph scheduling
Shorter response time
+
Less capacity loss
1. State of the art
2. Our approach
3. Future work
Graph-based architecture

Portability to diverse hardware

Does OpenVX really target “real-time” processing?

Source: https://www.khronos.org/openvx/
Does OpenVX really target “real-time” processing?

1. It lacks real-time concepts
2. Entire graphs = monolithic schedulable entities
Does OpenVX really target “real-time” processing?

1. It lacks real-time concepts
2. Entire graphs = monolithic schedulable entities
Does OpenVX really target “real-time” processing?

1. It lacks real-time concepts
2. Entire graphs = monolithic schedulable entities

Monolithic scheduling

Source: https://www.khronos.org/openvx/
Prior Work

Coarse-grained scheduling

- OpenVX nodes = schedulable entities \([23, 51]\)
Prior Work

Coarse-grained scheduling

• OpenVX nodes = schedulable entities [23, 51]

Remaining problems:

1. More parallelism to be explored
2. Suspension-oblivious analysis was applied and causes capacity loss.
Fine-Grained Scheduling

This Work
1. Coarse-grained vs. fine-grained
2. Response-time bounds analysis
3. Case study
1. Coarse-grained vs. fine-grained
2. Response-time bounds analysis
3. Case study
Coarse-Grained Scheduling

Task A:
Task B:
Task C:
Task D:
Suspension for GPU execution

Time

Fine-Grained Scheduling

Task A:
Task E:
Task F:
Task G:
Task C:
Task D:
1. Coarse-grained vs. fine-grained

2. Response-time bounds analysis

3. Case study
Deriving Response-Time Bounds for a DAG*

**Step 1:** Schedule the nodes as sporadic tasks

**Step 2:** Compute bounds for every node

**Step 3:** Sum the bounds of nodes on the critical path

Deriving Response-Time Bounds for a DAG
Deriving Response-Time Bounds for a DAG
Deriving Response-Time Bounds for a DAG

CPU

A

B

C

D

... 

GPU

E

... 

Need a response-time bound analysis for GPU tasks
A system model of GPU Tasks

\( \tau_i = (C_i, T_i, B_i, H_i) \)

- Per-block worst-case workload
- Number of blocks
- Period
- Number of threads per block (or block size)

\( \tau_1 = (3076, 6, 2, 1024) \)
Response-Time Bounds
Proof Sketch

1. We first show the necessity of a total utilization bound and intra-task parallelism via counterexamples.
Response-Time Bounds Proof Sketch

1. We first show the necessity of a total utilization bound and intra-task parallelism via counterexamples.
Response-Time Bounds
Proof Sketch

1. We first show the necessity of a total utilization bound and intra-task parallelism via counterexamples.

2. We then bound the unfinished workload from jobs released at or before $r_{k,j}$.

3. We prove the job finishes before $r_{k,j} + R_k$. 
1. Coarse-grained vs. fine-grained
2. Response-time bounds analysis
3. Case study
Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling

- **Application:** Histogram of Oriented Gradients (HOG)
Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling

- **Application**: Histogram of Oriented Gradients (HOG)
  - 6 instances
  - 33 ms period
  - 30,000 samples

- **Platform**: NVIDIA Titan V GPU + Two eight-core Intel CPUs.

- **Schedulers**: G-EDF, G-FL (fair-lateness)
Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling

- 50% samples have response time less than 60 ms
- Left is better
Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Response Time (ms)</td>
<td>65.99</td>
<td>136.57</td>
<td>84669.47</td>
</tr>
<tr>
<td>Maximum Response Time (ms)</td>
<td>125.66</td>
<td>427.07</td>
<td>170091.06</td>
</tr>
</tbody>
</table>

FL: fair-lateness
Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling

<table>
<thead>
<tr>
<th>Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Response Time (ms)</td>
</tr>
<tr>
<td>Maximum Response Time (ms)</td>
</tr>
</tbody>
</table>
Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling

<table>
<thead>
<tr>
<th>Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Response Time (ms)</td>
</tr>
<tr>
<td>Maximum Response Time (ms)</td>
</tr>
</tbody>
</table>

Half the average response time

One-third the maximum response time
Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Response Time (ms)</td>
<td>65.99</td>
<td>136.57</td>
<td>84669.47</td>
</tr>
<tr>
<td>Maximum Response Time (ms)</td>
<td>125.66</td>
<td>427.07</td>
<td>170091.06</td>
</tr>
</tbody>
</table>
Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Response Time (ms)</td>
<td>65.99</td>
<td>136.57</td>
<td>84669.47</td>
</tr>
<tr>
<td>Maximum Response Time (ms)</td>
<td>125.66</td>
<td>427.07</td>
<td>170091.06</td>
</tr>
</tbody>
</table>

FL: fair-lateness

Half the average response time

One-third the maximum response time
Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Response Time (ms)</td>
<td>65.99</td>
<td>136.57</td>
<td>84669.47</td>
</tr>
<tr>
<td>Maximum Response Time (ms)</td>
<td>125.66</td>
<td>427.07</td>
<td>170091.06</td>
</tr>
<tr>
<td>Analytical Bound (ms)</td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
</tbody>
</table>
Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Response Time (ms)</td>
<td>65.99</td>
<td>136.57</td>
<td>84669.47</td>
</tr>
<tr>
<td>Maximum Response Time (ms)</td>
<td>125.66</td>
<td>427.07</td>
<td>170091.06</td>
</tr>
<tr>
<td>Analytical Bound (ms)</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Response Time (ms)</td>
<td>65.99</td>
<td>136.57</td>
<td>84669.47</td>
</tr>
<tr>
<td>Maximum Response Time (ms)</td>
<td>125.66</td>
<td>427.07</td>
<td>170091.06</td>
</tr>
<tr>
<td>Analytical Bound (ms)</td>
<td>542.39</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

FL: fair-lateness
Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling

An alert driver takes 700 ms to react.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Response Time (ms)</td>
<td>65.99</td>
<td>136.57</td>
<td>84669.47</td>
</tr>
<tr>
<td>Maximum Response Time (ms)</td>
<td>125.66</td>
<td>427.07</td>
<td>170091.06</td>
</tr>
<tr>
<td>Analytical Bound (ms)</td>
<td>542.39</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Case Study: Comparing Fine-Grained/Coarse-Grained/Monolithic Scheduling

- Fair-lateness-based scheduler is beneficial as it reduced node response times by up to 9.9%.
- Overheads of supporting fine-grained scheduling was 14.15%.
Conclusions

1. Fine-grained scheduling

2. Response-time bounds analysis for GPU tasks

3. Case study
Future Work

1. Cycles in the graph
2. Other resource constraints
3. Schedulability studies
Thanks!