Efficient Compilation of Stream Programs for Heterogeneous Architectures
A Model-Checking based approach

Rajesh Kumar Thakur    Y. N. Srikant

Dept. of Computer Science and Automation,
Indian Institute of Science,
Bangalore-560012, INDIA

June 2, 2015
Agenda of the Talk

1. Motivation
2. Stream Programming Model
   - Preprocessing Stream Programs
3. Compilation Flow for Stream Programs
4. Overview of Model Checking
5. Execution Architecture Configuration
6. Building Computation Models from Stream Graph
   - Task Parallelism
   - Data Parallelism
   - Modelling Data Parallelism with Integrated Fission
   - State Space and Reachability Property
7. Modulo Scheduled Code Generation
8. Experimental Evaluation
9. Conclusion
Motivation

- Large number of application fit to Stream Programming model.
  - Multimedia, Graphics, Cryptography etc.
Motivation

- Large number of application fit to Stream Programming model.
  - Multimedia, Graphics, Cryptography etc.
- Stream programs can be represented as structured graphs, have regular and repeating computation, with explicit communication.
Large number of applications fit to Stream Programming model. Multimedia, Graphics, Cryptography etc.

Stream programs can be represented as structured graphs, have regular and repeating computation, with explicit communication.

Stream Program exposes data, task and pipeline parallelism.
Large number of application fit to Stream Programming model.
- Multimedia, Graphics, Cryptography etc.

Stream programs can be represented as structured graphs, have regular and repeating computation, with explicit communication.

Stream Program exposes data, task and pipeline parallelism.

Heterogeneous architectures are to become mainstream and hence it is challenging to obtain efficient compilation and execution of programs onto these architectures.
A stream graph $G = \{V, E\}$, where $V = \{v_1, \ldots, v_n\}$ is the set of actors/filters, and $E \subseteq V \times V$ is the set of FIFO communication channels between actors. A channel $(v_i, v_j) \in E$ buffers tokens (data elements) which are passed from the output of $v_i$ to the input of $v_j$. Synchronous dataflow (SDF) restricts the model by fixing the number of input and output tokens of a actor $v_i$. 

Rajesh Thakur, IISc, Bangalore. Efficient Compilation of Stream Programs for Heterogeneous Architectures, SCOPES'15, 7.
The actors here are of types: stateless, identity, and stateful.

Identity actors if fused can be efficiently replaced with a single actor performing the same task, with no visible loss of performance.
Compilation Flow for Stream Programs

1. Select Actors for Profiling.
2. Profile Results (Computation and Communication time).
3. CPU and GPU Execution Configuration selection.
4. Automata modelling pipeline, data, and task parallelism.
5. Solving least cost path reachability problem (Schedule).
6. Modulo Scheduling for Concerted Execution of Stream Program on CPU and GPU.
Overview of Model Checking

- Safety Property.
- Reachability property.
- Liveness Property.
Our compilation target is a heterogeneous combination of cores with different ISA (Instruction Set Architecture) and address space.
- Including both multicore CPUs and NVIDIA GPUs.
Our compilation target is a heterogeneous combination of cores with different ISA (Instruction Set Architecture) and address space.

- Including both multicore CPUs and NVIDIA GPUs.

NVIDIA GPUs provides 1000’s of computational cores. In order to overlap computation and communication, CUDA permits execution of programs in several stages, called *streams*.
Our compilation target is a heterogeneous combination of cores with different ISA (instruction Set Architecture) and address space.

- Including both multicore CPUs and NVIDIA GPUs.

NVIDIA GPUs provides 1000’s of computational cores. In order to overlap computation and communication, CUDA permits execution of programs in several stages, called *streams*.

*Streams* are defined as a sequence of operations that are performed in order on the device.

- A Stream sequence contains:
  - one memory copy from host (CPU) to device (GPU), which transfers input data;
  - one kernel launch, which uses this input data;
  - one memory copy from device to host, which transfers results.

- Lets the programmers launch multiple kernels onto GPUs, which facilitates execution of multiple actors onto GPUs in parallel.
Here, we assume we have two CPU cores (M1 and M2) and one GPU G1.

\[
\begin{align*}
\text{QM1} & \quad Q = 1 \\
\quad & \text{cost} \mathbin{+}= 6 \\
\text{Q'M1} & \quad Q = 0 \\
\quad & \text{S1 = 1} \\
\text{S1M1} & \quad S1 = 1 \\
\quad & \text{cost} \mathbin{+}= 6 \\
\text{S1'M1} & \quad S1 = 0 \\
\quad & \text{S2 = F3 = 1} \\
\text{S1M2} & \quad S1 = 1 \\
\quad & \text{cost} \mathbin{+}= 6 \\
\text{S1'G1} & \quad S1 = 0 \\
\quad & \text{S2 = F3 = 1} \\
\end{align*}
\]
The *channel* here is not the communication channel of the stream graph.

\[ F_{12} = 0 \]
\[ J_1 = 1 \]
\[ \text{cost} += c_1 \]
\[ t_{\text{tran}} = 1 \]

\[ F_{3} = 0 \]
\[ \text{cost} += c_2 \]
\[ F_{3} = 0 \]
\[ t_{\text{tran}} = 1 \]

\[ F_{4} = 0 \]
\[ \text{cost} += c_3 \]
\[ F_{4} = 0 \]
\[ J_1 = 1 \]
Modelling Data Parallelism

Rajesh Thakur, IISc, Bangalore. Efficient Compilation of Stream Programs for Heterogeneous Architectures, SCOPES'15, 16
The filters are fused and the model is build with all the possible fused combination of the filters (i.e. F41, F41F42, F41F42F43)
Reachability property $E \leftrightarrow (\text{FinalState} \text{ and } cost < \infty)$ is to be verified. The result is that the property is satisfied. Below is the trace obtained from UPPAAL.
Processor and Stage Assignment

Processes
- M1
- G1
- M2

Stages
- Stage 0
- Stage 1
- Stage 2
- Stage 3
- Stage 4

DMA

Rajesh Thakur, IISc, Bangalore. Efficient Compilation of Stream Programs for Heterogeneous Architectures, SCOPES'15, 19
Modulo Scheduled Code Generation
### Benchmarks Characteristics

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Total</th>
<th>Stateful</th>
<th>Peeking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>82</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BitR</td>
<td>452</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>CV</td>
<td>54</td>
<td>2</td>
<td>34</td>
</tr>
<tr>
<td>DCT</td>
<td>22</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td>DES</td>
<td>375</td>
<td>180</td>
<td>1</td>
</tr>
<tr>
<td>FFT-C</td>
<td>26</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>FFT-F</td>
<td>99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FB</td>
<td>53</td>
<td>34</td>
<td>16</td>
</tr>
<tr>
<td>FM-R</td>
<td>67</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>MM</td>
<td>52</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>MPEG</td>
<td>39</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>TDE</td>
<td>55</td>
<td>27</td>
<td>2</td>
</tr>
</tbody>
</table>
Makespan of Stream Programs

The time taken to complete a single iteration of the SDF graph is called the *makespan*.

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Makespan (ns)</th>
<th>MC-SWP</th>
<th>Malik et.al.</th>
<th>Udupa et.al.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biti</td>
<td>72570</td>
<td>77202</td>
<td>84292</td>
<td></td>
</tr>
<tr>
<td>BitR</td>
<td>105262</td>
<td>116958</td>
<td>147102</td>
<td></td>
</tr>
<tr>
<td>CV</td>
<td>8587960</td>
<td>8853568</td>
<td>10373877</td>
<td></td>
</tr>
<tr>
<td>DCT</td>
<td>1524609</td>
<td>1621925</td>
<td>1787428</td>
<td></td>
</tr>
<tr>
<td>DES</td>
<td>371921</td>
<td>413246</td>
<td>464369</td>
<td></td>
</tr>
<tr>
<td>FFT-C</td>
<td>317839</td>
<td>327669</td>
<td>413723</td>
<td></td>
</tr>
<tr>
<td>FFT-F</td>
<td>394579</td>
<td>419765</td>
<td>454031</td>
<td></td>
</tr>
<tr>
<td>FB</td>
<td>636420</td>
<td>707133</td>
<td>801904</td>
<td></td>
</tr>
<tr>
<td>FM-R</td>
<td>199727</td>
<td>205905</td>
<td>222543</td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td>1197292</td>
<td>1273715</td>
<td>1455675</td>
<td></td>
</tr>
<tr>
<td>MPEG</td>
<td>1675072</td>
<td>1861191</td>
<td>2033879</td>
<td></td>
</tr>
<tr>
<td>TDE</td>
<td>14065412</td>
<td>14500425</td>
<td>16111583</td>
<td></td>
</tr>
</tbody>
</table>
Performance Evaluation on Multicore CPU (Udupa et.al. vs MC-SWP)

MC-SWP : Max Speedup = 8.25X, Geometric Mean Speedup = 4.67X.
Udupa et.al. : Max Speedup = 7.77X, Geometric Mean Speedup = 2.93X.


Rajesh Thakur, IISc, Bangalore. Efficient Compilation of Stream Programs for Heterogeneous Architectures, SCOPES'15, 23
Performance Evaluation on Multicore CPU (Malik et.al. vs MC-SWP)

MC-SWP : Max Speedup = 8.25X, Geometric Mean Speedup = 4.67X.
Malik et.al. : Max Speedup = 8.17X, Geometric Mean Speedup = 4.68X.


Rajesh Thakur, IISc, Bangalore. Efficient Compilation of Stream Programs for Heterogeneous Architectures, SCOPES’15, 24
Performance Evaluation on Multicore CPU (All)

MC-SWP : Max Speedup = 8.25X, Geometric Mean Speedup = 4.67X.
Udupa et.al. : Max Speedup = 7.77X, Geometric Mean Speedup = 2.93X.
Malik et.al. : Max Speedup = 8.17X, Geometric Mean Speedup = 4.68X.

CPU : 2.4 Ghz 12 core Xeon E5646, 16 GB RAM

Rajesh Thakur, IISc, Bangalore. Efficient Compilation of Stream Programs for Heterogeneous Architectures, SCOPES’15, 25
Performance Evaluation on CPU (4 Cores) + GPU

MC-SWP : Max Speedup = 55.86X, Geometric Mean Speedup = 9.62X.
Udupa et.al. : Max Speedup = 49.32X, Geometric Mean Speedup = 6.76X.

CPU : 2.4 Ghz 12 core Xeon E5646, 16 GB RAM
GPU : NVIDIA TESLA 2075, 448 CUDA cores (14 Multiprocessors 32 CUDA Cores/MP)
Speedup comparison without streams (serial) and MC-SWP

- coalescing/decoalescing enabled.
Conclusion

We present a model-checking based framework for statically scheduling stream programs on heterogeneous architecture having both CPUs and GPUs.

Rajesh Thakur, IISc, Bangalore. Efficient Compilation of Stream Programs for Heterogeneous Architectures, SCOPES’15, 28
We present a model-checking based framework for statically scheduling stream programs on heterogeneous architecture having both CPUs and GPUs.

We produce a schedule which provides an efficient mapping onto these architectures and fully utilises the available resources.
We present a model-checking based framework for statically scheduling stream programs on heterogenous architecture having both CPUs and GPUs.

We produce a schedule which provides an efficient mapping onto these architectures and fully utilises the available resources.

We use CUDA streams on NVIDIA GPUs, where the optimal number of streams is decided using a profile-based approach.
Conclusion

- We present a model-checking based framework for statically scheduling stream programs on heterogenous architecture having both CPUs and GPUs.
- We produce a schedule which provides an efficient mapping onto these architectures and fully utilises the available resources.
- We use CUDA streams on NVIDIA GPUs, where the optimal number of streams is decided using a profile-based approach.
- To best of our knowledge our approach is the first one which utilises model-checking in a compiler that schedules and generates code for heterogeneous architectures.
We present a model-checking based framework for statically scheduling stream programs on heterogeneous architecture having both CPUs and GPUs.

We produce a schedule which provides an efficient mapping onto these architectures and fully utilises the available resources.

We use CUDA streams on NVIDIA GPUs, where the optimal number of streams is decided using a profile-based approach.

To best of our knowledge our approach is the first one which utilises model-checking in a compiler that schedules and generates code for heterogeneous architectures.

Our approach provides a speedup of up to 55.86X and a geometric mean speedup of 9.62X over a single threaded CPU on StreamIt benchmarks.
THANK YOU.

Questions?

We would like to thank IMPECS for the support towards this project.