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

Apr 28-29, 2106

Published at 18th International Workshop on Software and Compilers for Embedded Systems (SCOPES) 2015.

1. Motivation
2. Stream Programming Model
   - Synchronous Data Flow (SDF)
3. Compilation Flow for Stream Programs
4. Building Computation Models from Stream Graph
   - Task Parallelism
   - Data Parallelism
5. State Space and Reachability Property
6. Processor and Stage Assignment
7. 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.

Motivation

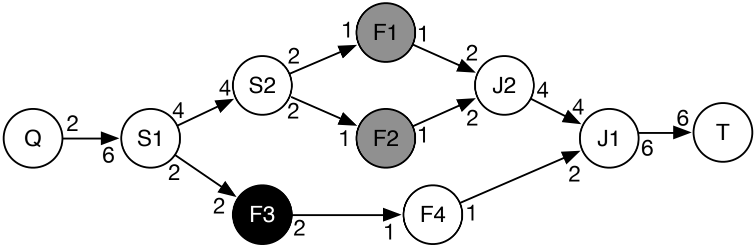
- Large number of application fit to Stream Programming model.
  - Multimedia, Graphics, Cryptography etc.
- Stream programs can be represented as structured graphs, have regular and repeating computation, with explicit communication.
- Stream Program exposes data, task and pipeline parallelism.

Stream Programming Model

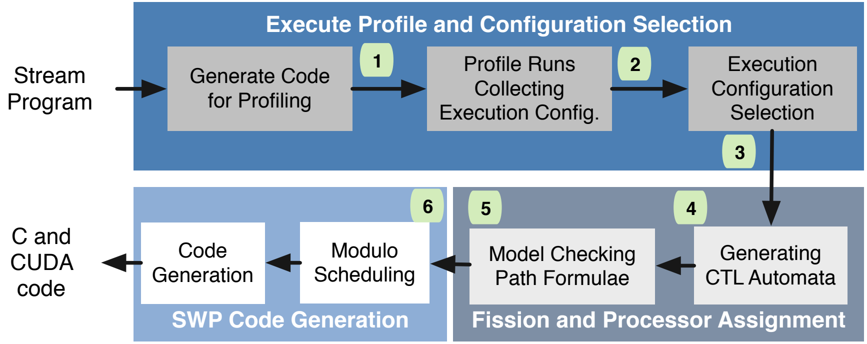
A stream graph $G = \{V, E\}$, where $V = \{v_1, ..., v_n\}$ is the set of actors/filters, and $E \subseteq V \times V$ is the set of FIFO communication channels between actors.

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 filter v_i.



Compilation Flow for Stream Programs


Execute Profile and Configuration Selection
Stream Program
Generate Code for Profiling
1
Profile Runs Collecting Execution Config.
2
Execution Configuration Selection
3
4
Generating CTL Automata
5
Model Checking Path Formulae
Fission and Processor Assignment
6
Modulo Scheduling
Code Generation
SWP Code Generation
C and CUDA code

1 **Select Actors for Profiling.** 2 **Profile Results (Computation and Communication time).**

3 **CPU and GPU Execution Configuration selection.**

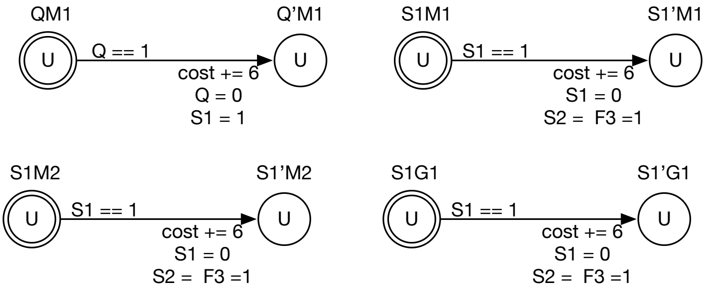
4 **Automata modelling pipeline, data, and task parallelism.**

5 **Solving least cost path reachability problem (Schedule).**

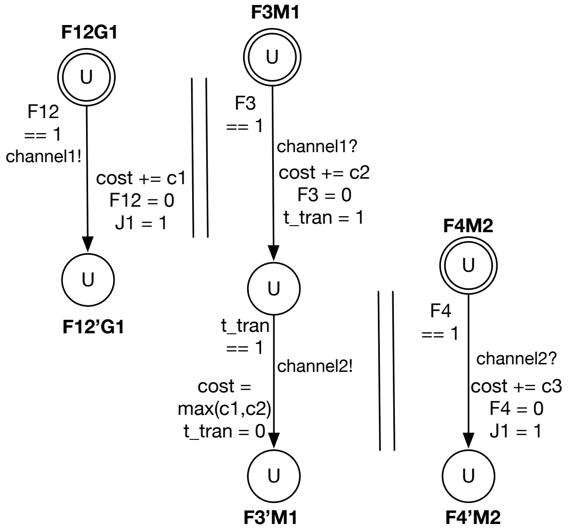
6 **Modulo Scheduling for Concerted Execution of Stream Program on CPU and GPU.**

Building Computation Models from Stream Graph

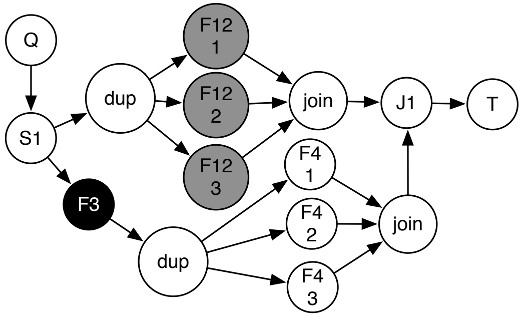
Our compilation target is a heterogeneous combination of cores with different ISA(instruction Set Architecture) and address space. Assuming two CPU cores (M1 and M2) and one GPU G1.



Modelling Task Parallelism



Modelling Data Parallelism



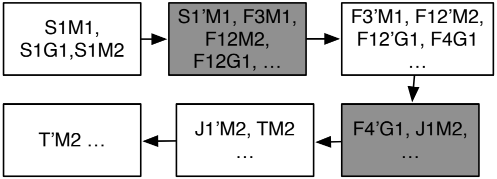
Data Parallelism modeling also incorporates integrated fission.

State Space and Reachability Property

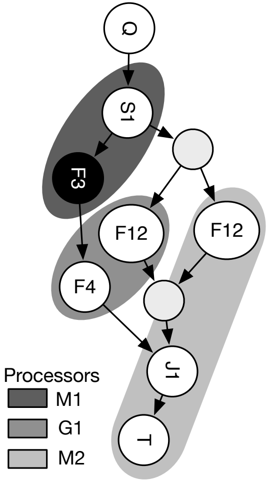
Reachability property

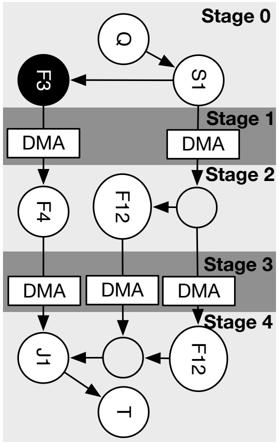
$$E <> (FinalState \ and \ cost < \infty)$$

is to be verified.
Below is the trace obtained from UPPAAL.



Processor and Stage Assignment

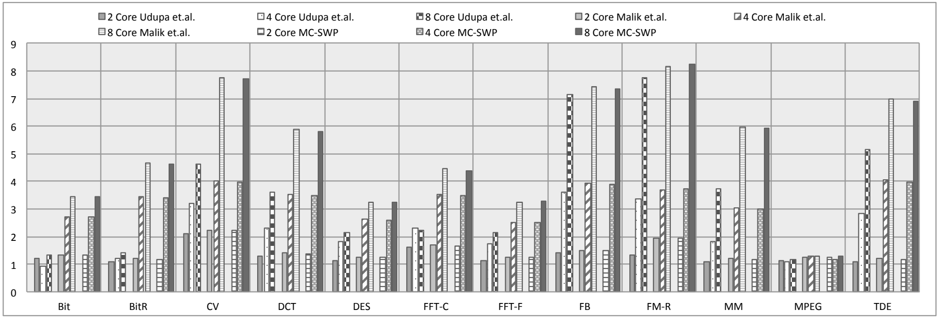




Modulo Scheduled code is the generated from the obtained stage

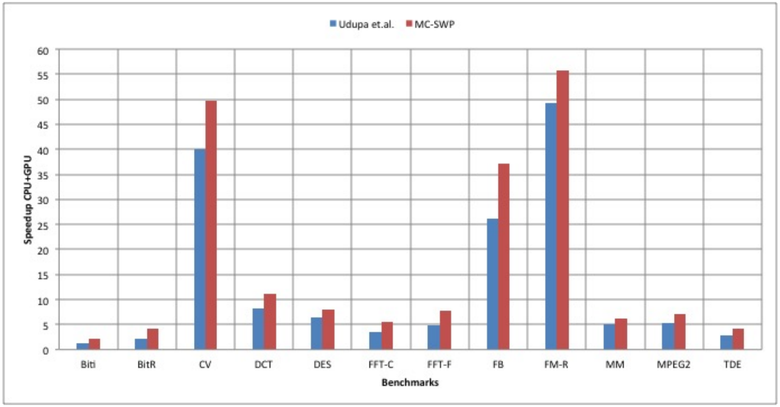
Performance Evaluation on Multicore CPU (All)

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



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.



Conclusion

- We present a model- checking based framework for statically scheduling stream programs on heterogenous architectures having both CPU and GPUs. (Our approach is the first one which utilises model-checking).

Conclusion

- We present a model- checking based framework for statically scheduling stream programs on heterogenous architectures having both CPU and GPUs. (Our approach is the first one which utilises model-checking).
- We produce a schedule which provides an efficient mapping onto these architectures and fully utilises the available resources.

Conclusion

- We present a model- checking based framework for statically scheduling stream programs on heterogenous architectures having both CPU and GPUs. (Our approach is the first one which utilises model-checking).
- 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 architectures having both CPU and GPUs. (Our approach is the first one which utilises model-checking).
- 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 architectures having both CPU and GPUs. (Our approach is the first one which utilises model-checking).
- 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.
- Our approach provides a speedup of upto 55.86X and a geometric mean speedup of 9.62X over a single threaded CPU on StreamIt benchmarks.

Thank You.
Questions?