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, IISc Bangalore
rajesh@csa.iisc.ernet.in

Motivation

1. Large number of application fit to Stream Programming model.
   (a) Multimedia, Graphics, Cryptography etc.
2. Stream programs can be represented as structured graphs, have regular and repeating computation, with explicit communication.
3. Stream Program exposes data, task and pipeline parallelism.
4. Heterogeneous architectures are to become mainstream and hence it is challenging to obtain efficient compilation and execution of programs onto these architectures.

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.

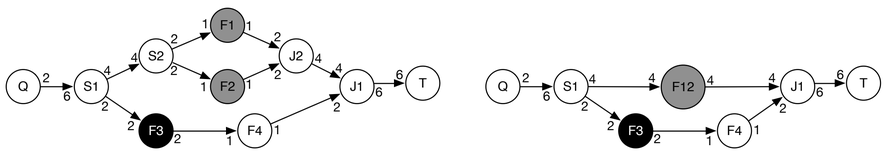


Figure 1: SDF Stream Graph and its Modified Graph

Compilation Flow for Stream Programs

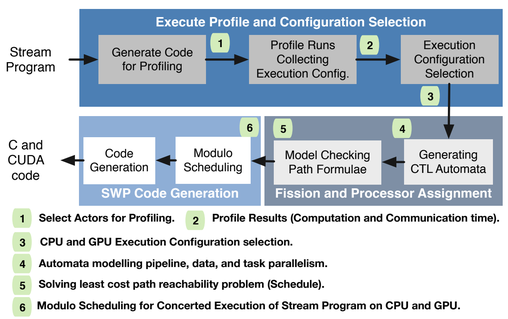
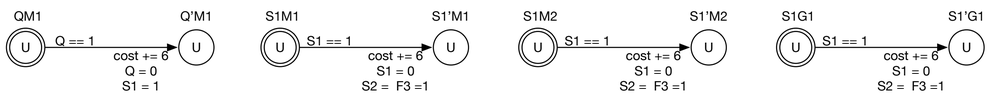


Figure 2: Compilation Flow

Building Computation Models from Stream Graph

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



Modeling Data and Task Parallelism and Integrated Fission

Assuming two CPU cores (M1 and M2) and one GPU G1.

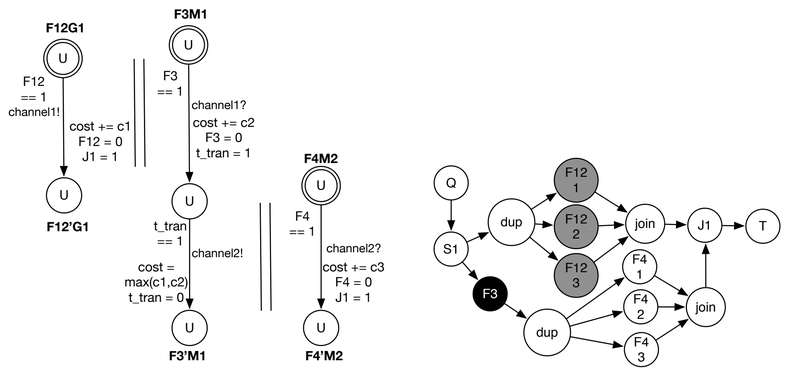


Figure 3: Task and Data Parallelism

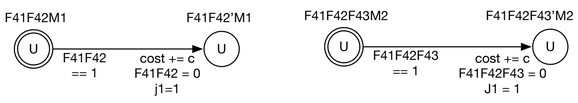


Figure 4: Optimal data parallelism exploitation

State Space and Reachability Property

Reachability property $E <> (FinalState\ and\ cost < \infty)$ is to be verified. A trace is then obtained from UPPAAL model checker.

Publication

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

Processor, Stage Assignment and Modulo Scheduled Code Generation

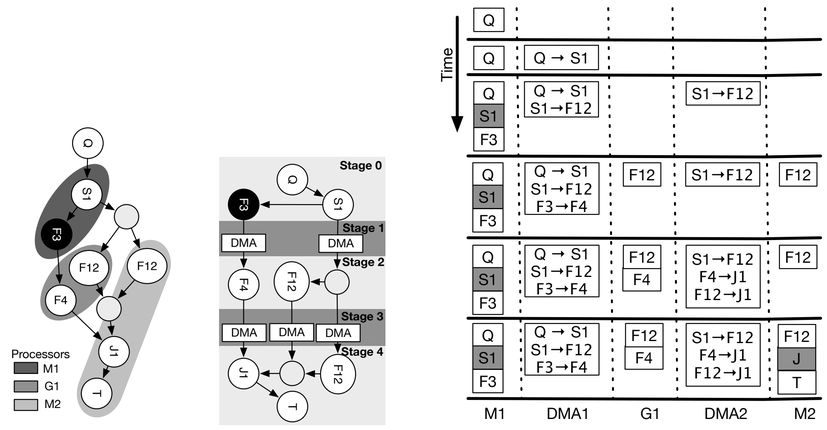


Figure 5: Processor Assignment, Stage Assignment, Modulo Scheduled Execution

Experimental Evaluation

Benchmarks	Actors		
	Total	Stateful	Peeking
Bit	82	0	0
BitR	452	2	0
CV	54	2	34
DCT	22	18	16
DES	375	180	1
FFT-C	26	14	0
FFT-F	99	0	0
FB	53	34	16
FM-R	67	23	22
MM	52	2	0
MPEG	39	7	0
TDE	55	27	2

Table 1: Characteristics of the Benchmarks

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

Table 2: Makespan

Performance Evaluation Results

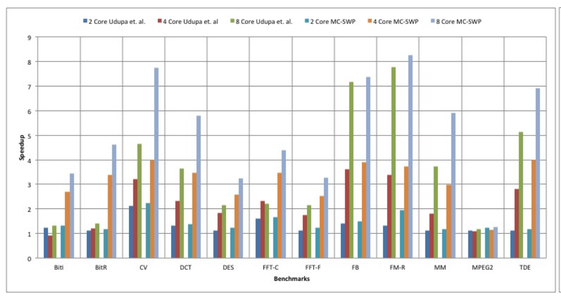


Figure 6: Performance Evaluation on Multicore CPU (Udupa et.al. vs MC-SWP)

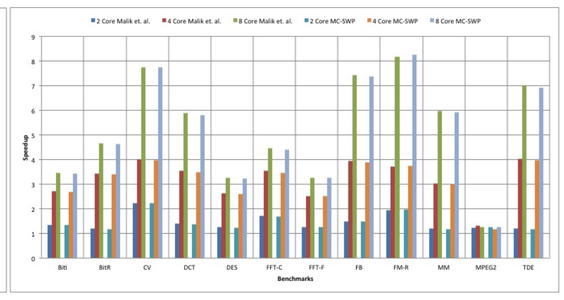


Figure 7: Performance Evaluation on Multicore CPU (Malik et.al. vs MC-SWP)

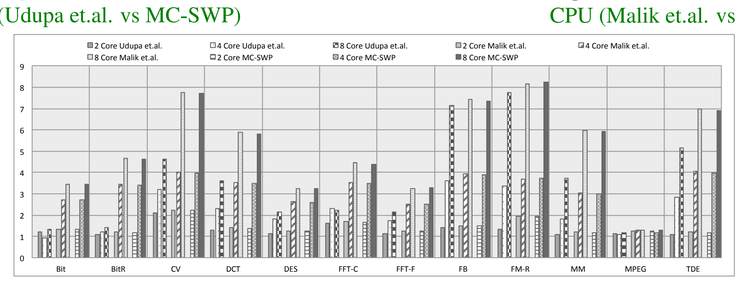


Figure 8: Performance Evaluation on Multicore CPU (ALL)

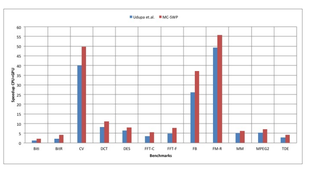


Figure 9: Performance Evaluation on CPU (4 Cores) + GPU)

Conclusions

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

Forthcoming Research

Scheduling programmable streams from imperative object-oriented programming languages onto CPU and GPUs, without programmer's intervention, preserving the execution semantics as specified in the language specification onto heterogeneous architectures.

Acknowledgements

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