Scheduling Parametric Data Flow Graphs

Vagelis Bebelis
vagelis.bebelis@inria.fr

INRIA - STMircoelectronics

SYNCHRON 2012

Alain Girault (INRIA)
Pascal Fradet (INRIA)
Bruno Lavigueur (STM)

Introduction

Goal:

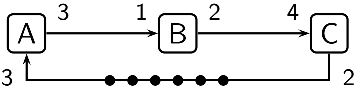
Scheduling parametric data flow applications on many core platforms

- Background: Data flow models, from synchronous to parametric
- Many-Core platform: Platform 2012
- Scheduling: Scheduling framework for parametric data flow
- Perspective: Possible future work and exploration

Outline

Synchronous Data Flow

- Synchronous Data Flow [1] (SDF): Each port has a fixed rate known at compile time.

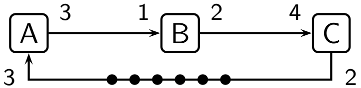


A Synchronous Data Flow graph

- Actors: Function units
- Edges: Communication links (FIFO)
- Port rate: Number of tokens transferred through a port
- Graph State: Number of tokens on the graph's edges
  $S_i = \begin{bmatrix} 0 & 0 & 6 \end{bmatrix}$

[1] Synchronous Data Flow, E.A.Lee et al. 1987

Synchronous Data Flow



SDF graph

- Balance equations: $$\begin{aligned} \#A \cdot 3 &= \#B \\ \#B \cdot 2 &= \#C \cdot 4 \\ \#C \cdot 2 &= \#A \cdot 3 \end{aligned}$$
- Repetition vector: $\begin{bmatrix} 2 & 6 & 3 \end{bmatrix}$
- Iteration: Sequence of firings that return the graph to the initial state
- Schedule: Execution of a complete iteration
  e.g. Single Appearance Schedule: A^2, B^6, C^3
- Liveness: enough initial tokens to fire actors on a directed cycle

Advantages & Disadvantages of SDF

Advantages

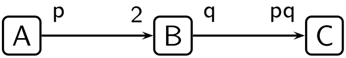
+ Natural expression of DSP applications
+ Finite memory - Boundedness guarantee
+ Deadlock-free operation - Liveness guarantee
+ Static scheduling - Timing guarantee

Disadvantages

– Not expressive enough
  (e.g. for video codec applications)

Parametric Data Flow

- Parametric Data Flow (PDF) uses parameterized instead of fixed port rates.



A parametric data flow graph

- Simplified version of SPDF [2] and PSDF [3] models
- No parameter changes within iterations
- No hierarchical structures
- Symbolic analysis of the graph
  Repetition vector: $\begin{bmatrix} 2 & p & 1 \end{bmatrix}$

[2] Schedulable Parametric Data Flow, P.Fradet et al. 2012

[3] Parametric Dataflow Modelling for DSP Systems, B.Bhattacharya et al. 2001

Advantages & Disadvantages of PDF

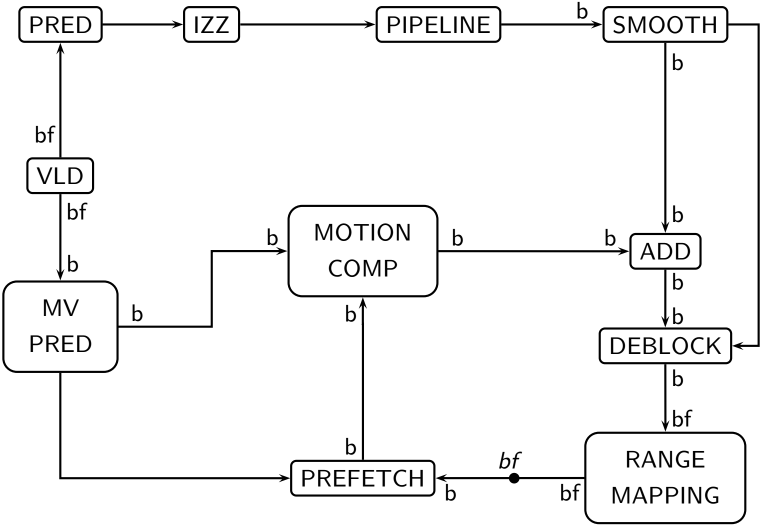
Advantages

+ Finite memory - Boundedness guarantee
+ Deadlock-free operation - Liveness guarantee
+ Expression of video applications

Disadvantages

– Static scheduling possible but too restrictive
  (e.g. As soon as possible schedule cannot always be expressed)

Example: VC-1 decoder capture in PDF



Outline

Platform 2012

Platform Features

- Many - core platform designed by STMicroelectronics
- 1-32 clusters with 1-16 cores:
  - Software cores: General Purpose Processors (GPP)
  - Hardware cores: Hardware processing elementes (HWPE)
- Native programming model

Mapping assumptions

- Application fits in a single cluster
- Each actor is executed by a GPP or implemented as a HWPE
- The schedule is executed by a GPP

Native programming model

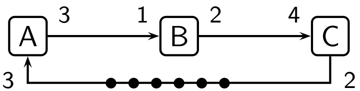
- Predicated Execution Data Flow (PEDF) model
- Simplifies the parallel implementation of applications

- Data flow implementation of applications:
  - Filter: Basic functional block
  - Controller: Schedules the firing of filters and controls the configuration parameters

- Uses a slot notion for scheduling like in blocked scheduling [4]
  + All filters synchronize after each cycle of iterations
  + Reduces computational complexity of parallel scheduling
  + Compatible with other many-core platforms (CUDA, OpenGL)
  − Introduces slack

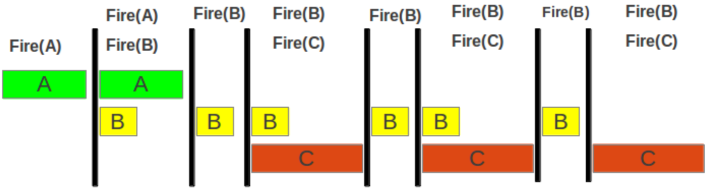
[4] Compile-time Scheduling and Assignment of Data-flow Program Graphs with Data-Dependent Iteration, S.Ha et al. 1991

Scheduling example



SDF graph

Repetition vector: $\begin{bmatrix}2 & 6 & 3\end{bmatrix}$



Outline

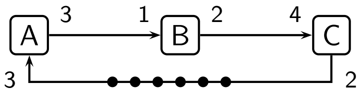
1. Data Flow Models
2. Platform 2012
3. Scheduling
   - Scheduling SDF graphs
   - Scheduling PDF graphs
   - Scheduling framework
4. Future Work

Scheduling SDF graphs

Scheduling goal

Schedule all firings to complete an iteration

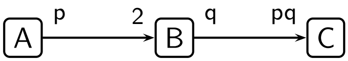
- Repetition vector: $\begin{bmatrix} 2 & 6 & 3 \end{bmatrix}$



SDF graph

- Scheduling examples:
  - **Sequential schedules:**
  - A^2, B^6, C^3 - Single appearance schedule
  - $A, B^2, C, B, A, B, C, B^2, C$ - Minimum buffer size schedule
  - **Parallel schedules:**
  - $A, (A \| B), [B, (B \| C)]^2, B, C$ - As Soon As Possible schedule (ASAP)

Scheduling PDF graphs

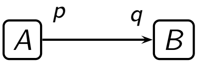


PDF graph

- Repetition vector: $\begin{bmatrix} 2 & p & 1 \end{bmatrix}$
- Single appearance schedule: A^2, B^p, C
- Quasi-static schedule uses parameters to express a schedule at compile time
- Unable to produce ASAP schedule (*next slide*)
- No scheduling flexibility

ASAP scheduling of PDF graphs

- Considering the generic PDF edge:



- Minimum firings of actor A for ASAP firing of actor B:

1. $n_1 = \left\lceil \frac{q}{p} \right\rceil$ and $r_1 = n_1 \cdot p - q$
2. $n_2 = \left\lceil \frac{q-r_1}{p} \right\rceil$ and $r_2 = n_2 \cdot p - q$
3. $n_3 = \left\lceil \frac{q-r_2}{p} \right\rceil$ and $r_3 = n_3 \cdot p - q$
4. $\ldots$

- As the number of remaining tokens change constantly a quasi-static schedule is not possible.

Scheduling framework features

Features

- As Soon As Possible schedule
  - Minimizes schedule span (no resource sharing)
  - Maximizes parallelism
- Flexible to be reused
  - Different platforms
  - Optimization criteria
  - Scheduling strategies

- Main idea: Produce different schedules while keeping the same algorithm
- Usage of scheduling constraints

Scheduling framework overview

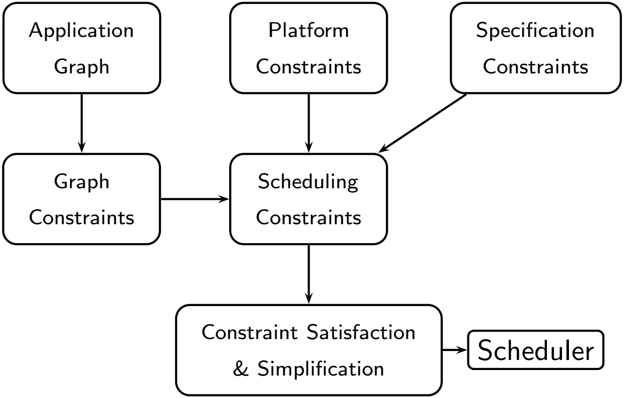


Figure: Scheduling framework

Scheduling constraints

A constraint is a relationship between the firings of two actors (X, Y):

$$\mathbf{X_i > Y_{f(i)}}$$

Interpretation

- The i_{th} firing of X waits for the $f(i)_{th}$ firing of Y
  $\Rightarrow X_i$ must be scheduled at a later slot than $Y_{f(i)}$

Constraint types

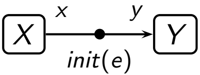
- **Graph constraints:** Data dependencies
- **Platform constraints:** Constraints due to platforms specificities
- **User constraints:** Constraints to optimize some criterion

Constraint Types

Graph Constraints

- Data dependencies derive from the graph.

- For each edge of the graph we get:

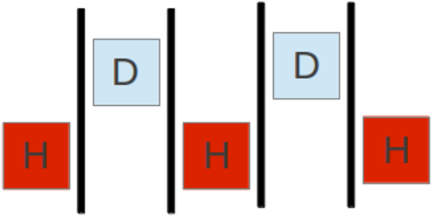


- $Y_i > X_{f(i)}$ with $f(i) = \left\lceil \frac{y \cdot i - init(e)}{x} \right\rceil$

Constraint Types

Platform Constraints

- Platform constraints are given and derive from platform specificities
- Applied to all application of the platform

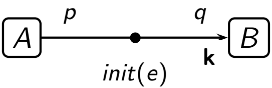


- Example:
  Prevention of power intensive filter *H* from firing twice in a row
  $\Rightarrow$ Introduction of dummy filter D
- $H_i > D_{i-1}$ and $D_i > H_i$

Constraint Types

User Constraints

- User constraints are used to achieve specific schedule behaviour
  - To express an optimization
  - To set a scheduling strategy



- Example:
  Buffer capacity restriction to k tokens
  $\Rightarrow A_i > B_{g(i)}$ with $g(i) = \left\lceil \frac{p \cdot i + init(e) - k}{q} \right\rceil$
- **May introduce deadlocks!!!** $\Rightarrow$ Constraint satisfaction algorithm to ensure compatibility

Constraint deadlock Detection

Deadlock

A set of constraints deadlocks when there is a constraint that propagates from an actor's firing to the same or future firing of the same actor.

$$\exists A, i, j, (A_i > A_j) \wedge (i \leq j)$$

- Thus for each cycle:

$$A_i > B_{f_1(i)} > \cdots > C_{f_{n-1}(i)} > A_{f_n(i)}$$

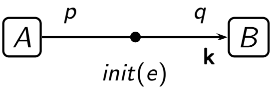
$$\Rightarrow A_i > A_{f_1(\cdots(f_{n-1}(f_n(i)))}$$

- Check if

$$i > f_1(\cdots(f_{n-1}(f_n(i)))$$

Deadlock detection example

- Considering the constraints between actors A and B:

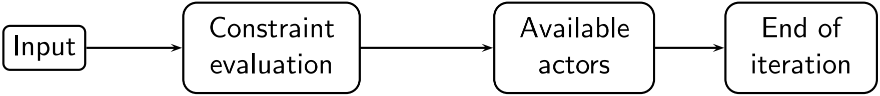


- Graph constraint: $B_i > A_{f(i)}$ with $f(i) = \left\lceil \frac{q \cdot i - init(e)}{p} \right\rceil$
- User constraint: $A_i > B_{g(i)}$ with $g(i) = \left\lceil \frac{p \cdot i + init(e) - k}{q} \right\rceil$
- Circular constraint: $A_i > A_{f(g(i))}$
- Deadlock check: $i > f(g(i)) \Rightarrow i > \left\lceil \frac{q \cdot \lceil \frac{p \cdot i + init(e) - k}{q} \rceil - init(e)}{p} \right\rceil$

Run-time scheduler

Input

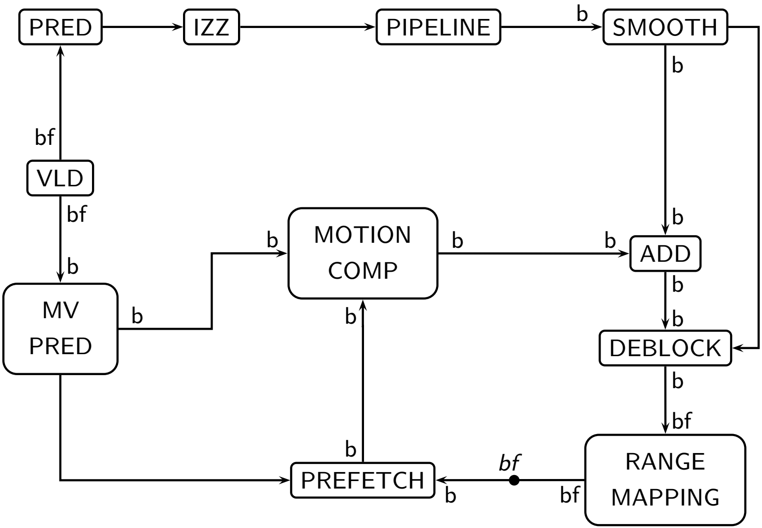
- Set of actors
- Repetition vector
- List of constraints



Overhead

- Overall small overhead:
  - Concurrent execution with graph actors
  - Small amount of constraints
  - Optimization of static parts of the graph

Example use case: VC-1 decoder



Outline

Future Work

Conclusions

- Flexible constraint framework for PDF graphs
- Modular way to change the schedule without changing the scheduling algorithm
- Ability to express platform specificities and scheduling strategies
- Compile time guarantees of schedule liveness

Future work

- Scheduler optimization:Solve all static and quasi-static actors at compile-time, run-time scheduling only when necessary
- Extension of the PDF model to include boolean parameters
- Extension of the constraints to express more interesting scheduling strategies
- Implementation and integration of the framework within ST's tool-chain for the platform