Parametric Dataflow Programming: Model of Computation and Many-Core Scheduling

Alain Girault (alain.girault@inria.fr)

The 9th Scheduling for Large Scale Systems Workshop, July 2014

Joint work with
Vagelis Bebelis & Pascal Fradet (INRIA)
Presentation Overview

1. Data Flow Models of Computation
2. Scheduling Framework
3. Experiments
4. Conclusions
Outline

1. Data Flow Models of Computation
   - Synchronous Data Flow
   - Boolean Parametric Data Flow

2. Scheduling Framework

3. Experiments

4. Conclusions
SDF - Synchronous Data Flow

An SDF graph

Firing of actor A:
- Consumes 1 token

Firing of actor A:
- Produces 3 tokens

\[ \#A \cdot 3 = \#B \cdot 2 \]
\[ \#B \cdot 1 = \#C \cdot 3 \]
\[ \#C \cdot 2 = \#A \cdot 1 \]

Schedule:
- Series of firings that complete one iteration
- Here: \[ [A \ 2 \ B \ 3 \ C] \]

Sequential Single Appearance
Sequential Minimum Buffer Size
Parallel ASAP

---

1E.A.Lee and D.G.Messerschmitt, Proc. of the IEEE, 1987
SDF - Synchronous Data Flow

In SDF all rates are \textbf{fixed and known} at compile time.

\[\begin{bmatrix}
A & = & 2 \\
B & = & 3 \\
C & = & 1
\end{bmatrix}\]

\begin{itemize}
  \item \textbf{Actor} (Functional unit)
  \item \textbf{Edge} (Communication link)
  \item Rate (Amount of data)
  \item Initial tokens
\end{itemize}

\cite{Lee1987}
Firing of actor $A$: Consumes 1 token

\[
\begin{bmatrix}
#A & = & 2 \\
#B & = & 3 \\
#C & = & 1
\end{bmatrix}
\]
Firing of actor $A$: Produces 3 tokens
SDF – Balance Equations

\[
\begin{bmatrix}
#A &= 2 \\
#B &= 3 \\
#C &= 1
\end{bmatrix}
\]

\[
[A^2 \ B^3 \ C]
\]

\[
\begin{align*}
#A \cdot 3 &= #B \cdot 2 \\
#B \cdot 1 &= #C \cdot 3 \\
#C \cdot 2 &= #A \cdot 1
\end{align*}
\]
**Schedule**: Series of firings that complete one iteration

Here: \([A^2 \ B^3 \ C]\)

Sequential Single Appearance

\(A^2; B^3; C\)

Sequential Minimum Buffer Size

\(A; B; A; B^2; C\)

Parallel ASAP

\(A; (A \parallel B); B^2; C\)
Parametric data flow models

Need for more expressive data flow models

- SDF is not expressive enough for complex applications.
- More expressive models are employed that use
  - Parametric rates
  - Dynamic graph topology
- Both features make scheduling more difficult.
- Such a model is Boolean Parametric Data Flow (BPDF)\(^1\).

\(^1\)V. Bebelis, P. Fradet, A. Girault and B. Lavigneuer, EMSOFT’13, 2013
Boolean Parametric Data Flow - BPDF

A BPDF graph

A → B → C → E
B → D

A parameter
¬a
a
a

A. GIRault (INRIA)
Boolean Parametric Data Flow - BPDF

- Modifier of boolean $a$
- Integer parameter $p$
- Change period of boolean $a$
- Boolean guard on BD
BPDF analysis: Balance Equations

\[ \#A \cdot p = \#B \]

\[
\begin{bmatrix}
\#A = 2 \\
\#B = 2p \\
\#C = p \\
\#D = 2p \\
\#E = 2p \\
\end{bmatrix}
\]

\[ \begin{bmatrix} A^2 & B^{2p} & C^p & D^{2p} & E^{2p} \end{bmatrix} \]

Parametric solution of balance equations
BPDF analysis: Balance Equations

$$a: false$$

Actor C fires despite being disconnected
BPDF analysis: Balance Equations

There are implicit boolean propagation links
BPDF - Actor firing

(1) Read boolean parameters
(2) Read data from connected inputs
(3) Set boolean parameters
(4) ... Compute ...
(5) Write data to connected outputs
BPDF - Actor firing

(1) Read boolean parameters
(2) Read data from connected inputs
(3) Set boolean parameters
(4) ... Compute ...
(5) Write data to connected outputs
BPDF - Actor firing

1. Read boolean parameters
2. Read data from connected inputs
3. Set boolean parameters
4. ... Compute ...
5. Write data to connected outputs
BPDF - Actor firing

1. Read boolean parameters
2. Read data from connected inputs
3. Set boolean parameters
4. ... Compute ...
5. Write data to connected outputs
BPDF - Actor firing

1. Read boolean parameters
2. Read data from connected inputs
3. Set boolean parameters
4. ... Compute ...
5. Write data to connected outputs
BPDF Scheduling - Integer parameters

Sequential Single Appearance
Sequential Minimum Buffer Size
Parallel ASAP

A^p \rightarrow q \rightarrow B

A^q; B^p
Difficult to express
Difficult to express

Needs iterative comparison of the values of p and q
Parallel ASAP schedule when $g = true$ \[ A^p; B \]
Parallel ASAP schedule when $g = false$ \[ A; (A \parallel B); A^{p-2} \]

Also needs constant checking of the boolean values
Outline

1. Data Flow Models of Computation

2. Scheduling Framework
   - STHORM platform
   - Scheduling framework

3. Experiments

4. Conclusions
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 Elements (HWPE)

Mapping assumptions

- Application fits in a single cluster
- Each actor is executed on a GPP or implemented as a HWPE
- The schedule is executed on a GPP
Slotted scheduling model

- Compatible with the scheduling model of STHORM.
- Uses a slot notion like in blocked scheduling
  + Actors synchronize after each execution
  + Reduces complexity of parallel scheduling
  + Compatible with other parallel programming models (CUDA, OpenGL)
- May introduce slack

\[ [A^2 B^6 C^3] \]

---

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Fire(A)</td>
<td>A</td>
<td>Fire(B)</td>
<td>Fire(B)</td>
<td>Fire(B)</td>
<td>Fire(B)</td>
<td>Fire(B)</td>
<td>Fire(B)</td>
<td>Fire(B)</td>
<td>Fire(B)</td>
<td>Fire(B)</td>
<td>Fire(B)</td>
<td>Fire(B)</td>
<td>Fire(B)</td>
<td>Fire(C)</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>

---

Scheduling framework features

The framework should

- Automatically produce ASAP schedules
  - Best strategy when timing is unknown
- Be expressive and flexible for different
  - Platforms
  - Optimization criteria
  - Scheduling strategies

Main idea: Production of different schedules with the same (ASAP) algorithm

---

Scheduling framework overview

- Application
- User-defined Constraints
- Ordering Constraints
- Resource Constraints
- Simplification & Liveness
- Evaluate Ordering
- Scheduler
- Filtering
- ASAP Schedule
Scheduling constraints

- **Ordering Constraints**: Express the partial ordering of the firings

  \[ X_i > Y_{f(i)} \]

  where \( X_i \) is the \( i \)th firing of actor \( X \)

- **Resource Constraints**: Control the parallel execution

  replace \( S_A \) by \( S_B \) if condition

  where \( S_A \) and \( S_B \) are subsets of actors such that \( S_A \supseteq S_B \neq \emptyset \)

  (used for filtering the preliminary schedule)
Graph constraint: Data dependency

\[ B_i > A_{f(i)} \quad \text{with} \quad f(i) = \left\lceil \frac{q \cdot i - t}{p} \right\rceil \]

because \( t + f(i) \cdot p \geq q \cdot i \iff f(i) \geq \frac{q \cdot i - t}{p} \)

Modifier to user constraint: Boolean dependency

\[ U_i > M_{f(i)} \quad \text{with} \quad f(i) = \pi_w \cdot \left\lfloor \frac{i - 1}{\pi_r} \right\rfloor + 1 \]

where \( \pi_r \) (resp. \( \pi_w \)) is the reading (resp. writing) period of \( U \) (resp. \( M \))
Buffer Constraint: Buffer capacity restriction to $k$

$$A_i > B_{g(i)} \quad \text{with} \quad g(i) = \left\lceil \frac{p \cdot i + t - k}{q} \right\rceil$$

Resource Constraint: Mutual exclusion of $A$ and $B$

replace $\{A, \ B\}$ by $\{A\}$
Constraint liveness condition

User ordering constraints may introduce deadlocks

\[ B_j > A_k \]

\[ A_i > B_j \]

\[ \Rightarrow A_i > A_k \]

Liveness condition:

\[ \forall \text{ cycle } A_i > A_k \]

we need \( i > k \)
Deadlock detection example

Constraints:

Application: \( B_i > A_f(i) \)
Buffer: \( A_i > B_g(i) \)
Cycle: \( A_i > A_f(g(i)) \)

Liveness condition:
\[ i > f(g(i)) \]

Solution:
\[
i > f(g(i)) \iff i > \left\lceil \frac{q \cdot \left\lceil \frac{p \cdot i - k}{q} \right\rceil}{p} \right\rceil
\]
\[
\iff i > \frac{q \cdot (\frac{p \cdot i - k}{q} + 1)}{p} + 1
\]
\[
\iff i > i + \frac{q - k}{p} + 1
\]
\[
\iff k > p + q
\]
\[
\iff k > p_{\text{max}} + q_{\text{max}}
\]
Constraint simplification

\[ [A \ B^{2p} \ C^{3p}] \]

Constraints

\[ B_i > A \left\lceil \frac{i}{2p} \right\rceil \]

\[ C_i > B \left\lfloor \frac{2i}{3} \right\rfloor \]

\[ C_i > A \left\lceil \frac{i}{3p} \right\rceil \]

\[ C_i > B \left\lfloor \frac{i}{3} \right\rfloor - 1 \]

ASAP Schedule: \( A; B; (B\|C)^{2p-1}; C^{p+1} \)
If simplification is not possible then a run-time scheduler is employed.

Small overhead:

- Concurrent execution with actors
- Coarse - grain graphs (small number of actors)
- Simplification of constraints at compile time
- Optimization of static parts of the graph
Outline

1. Data Flow Models of Computation
2. Scheduling Framework
3. Experiments
4. Conclusions
Scheduler Overhead

- Implementation of TNR (Temporal Noise Reduction) application on STHORM platform
- Overhead comparison between
  - Dynamic scheduler
  - Simplified scheduler
  - Manually optimized schedule

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles / frame</td>
<td>2.140.000</td>
<td>1.100.000</td>
<td>360.000</td>
<td>340.000</td>
</tr>
</tbody>
</table>

Schedule overhead for different schedules of TNR
Use Case: VC-1 decoder

Experiments

Inter pipeline
Intra pipeline

Repetition vector:
\[ \{A B^p C^{pq} D^p E^{pq} F^{pq} G^p\} \]
Buffer restriction

Token accumulation on edge $DG$

- Maximum buffer estimation: $\sim pq - p$
- Buffer restriction to: $q$

$$D_i > G \left\lceil \frac{q \cdot i - q}{q} \right\rceil$$

- Increase of 2% in total schedule duration

Repetition vector:

$$[A B^p C^{pq} D^p E^{pq} F^{pq} G^p]$$
When timing is known, the goal is to minimize slack.

- Clustering of actors D and G in the same slot with:
  - replace D, E by E if ¬fireable(G)
- Improvement of 15% in the schedule duration.

Repetition vector:

\[[A \: B^p \: C^{pq} \: D^p \: E^{pq} \: F^{pq} \: G^p]\]
Outline

1. Data Flow Models of Computation
2. Scheduling Framework
3. Experiments
4. Conclusions
Conclusions

We proposed a scheduling framework for BPDF applications that

- is flexible and modular through constraints
- is expressive to optimize the schedule
- automatically generates ASAP parallel schedules
- Statically guarantees the boundness and liveness of the produced schedule
Ongoing and future work

- Use the framework with a **non-slotted** scheduling model
- **Formalization** of the constraint simplification procedure
- Use the framework to **optimize bi-criteria** scheduling, specifically power consumption vs. throughput
- Design a **high-level language** to express scheduling policies that can be automatically compiled into constraints
- Implementation of BPDF within **Ptolemy II**
- **Formal comparison** between BPDF and SADF

---

4 Stuijk et al, IC-SAMOS’11
Thank you for your attention!

Questions?
Run-time scheduler

- Reading / Writing Periods
- Boolean Values
- Actors
- Ordering Constraints
- Constraint Filtering
- Evaluate Ordering
- Repetition Vector
- Resource Constraints
- Filterable Actors
- Filtering
- FIRE