An Empirical Evaluation of a Programming Model for Context-Dependent Real-Time Streaming Applications

ICCS 2015, ALCHEMY Workshop

Khanh Do, Stephane Louise, Albert Cohen

stephane.louise@cea.fr

CEA LIST/DACLE

June 1, 2015
Outline

1. DataFlow models and determinism
2. Limitations of today’s CSDF models
3. Proposition of an improved MoC
4. Conclusion and outlooks
DataFlow models and determinism

DataFlow concepts are old (Kahn’s original paper is from 1974), but are under a renew interest from embedded computing, with the emergence of manycores.
DataFlow concepts are old (Kahn’s original paper is from 1974), but are under a renew interest from embedded computing, with the emergence of manycores.

- DataFlow = data driven. The processing follows the data paths
- Data paths are explicit = no need for memory coherence
- Naturally distributed processing, fits well with distributed computing
- Hierarchical, and easy to design for
DataFlow concepts are old (Kahn’s original paper is from 1974), but are under a renew interest from embedded computing, with the emergence of manycores.

- DataFlow = data driven. The processing follows the data paths
- Data paths are explicit = no need for memory coherence
- Naturally distributed processing, fits well with distributed computing
- Hierarchical, and easy to design for

But there are some weaknesses:

- Not carefully done, they can deadlock
- They can run out of memory
- None of these are decidable before runtime
SDF and CSDF models (mid 90s) avoid these pitfalls:

- At the price of versatility: Fixed rhythms of consumption and productions
- At compile-time, models allow to check that a given application
  - Cannot deadlock
  - Runs in the intended memory limits

The limits of the model exist but are not binding for signal and video processing applications. For more sophisticated applications, education of software architecture designers may be a requirement. Nonetheless, this is not a difficulty w.r.t programming challengingly parallel applications.
Languages like Brook and StreamIt brought some useful features to the model

- The number of accessed token in one run of a filter can be different than the number of token consumed
- Special filters are defined for data-distribution only, without modifying the values in the streams
  - Splitters: takes an input channel and distributed it in a fixed way among $n$ outputs channels
  - Duplicaters: takes an input channel and duplicates the tokens among $n$ output channels
  - Joiners: takes $n$ inputs in a fixed way and aggregates them in a single output channel
  - plus Sources (I/O) and Sinks
Compilation process of CSDF graphs

The compilation process:

- One source code fits all (the compiler takes the Process Network as specified by the source code and provides the optimization for a given platform)
- Can verify good properties
- Can aggregate filters to match the underlying architecture granularity
- Can increase (or decrease by aggregation) the size of the pipeline to match throughput and parallelism constraints
- Can optimize networks of data-distribution processes to:
  - Create more efficient networks of data-distribution (i.e. the programmer implements naively but the compiler create optimized versions of graph which distribute data in a very efficient manner [Pablo Oliveira et Al.])
  - Create copy-less access to buffers in the cascade of filters [Loic Cudennec et Al.]
Compilation process of CSDF graphs

The compilation process:

- One source code fits all (the compiler takes the Process Network as specified by the source code and provides the optimization for a given platform)
- Can verify good properties
- Can aggregate filters to match the underlying architecture granularity
- Can increase (or decrease by aggregation) the size of the pipeline to match throughput and parallelism constraints
- Can optimize networks of data-distribution processes to:
  - Create more efficient networks of data-distribution (i.e. the programmer implements naively but the compiler create optimized versions of graph which distribute data in a very efficient manner [Pablo Oliveira et Al.])
  - Create copy-less access to buffers in the cascade of filters [Loic Cudennec et Al.]

This was implemented e.g. in the ΣC compiler for Kalray’s MPPA-256 manycore chip [Thierry Goubier et Al.].
Simple Example: software radio

An Empirical Evaluation of a Programming Model for Context-Dependent Real-Time Streaming Applications | DACLE Division | June 1, 2015 | 7
Limitations of today’s CSDF models

Perceived evolution in system architectures:

- More dynamic, more open: e.g. using the cloud, or external resources
- Less reliable: some elements (communication channels or processing resources) can be non-functioning or non-reachable
- Power management have more and more importance for future platforms: use of computing and communication resources is sparser and must be used wisely to protect battery life (e.g. on demand)
- Time constraints or real-time properties are more and more required
Limitations of today’s CSDF models

Perceived evolution in system architectures:

- More dynamic, more open: e.g. using the cloud, or external resources
- Less reliable: some elements (communication channels or processing resources) can be non-functioning or non-reachable
- Power management have more and more importance for future platforms: use of computing and communication resources is sparser and must be used wisely to protect battery life (e.g. on demand)
- Time constraints or real-time properties are more and more required

This makes future platforms more dynamic in their processing power.
Limitations of today’s CSDF models

Perceived evolution in system architectures:

- More dynamic, more open: e.g. using the cloud, or external resources
- Less reliable: some elements (communication channels or processing resources) can be non-functioning or non-reachable
- Power management have more and more importance for future platforms: use of computing and communication resources is sparser and must be used wisely to protect battery life (e.g. on demand)
- Time constraints or real-time properties are more and more required

This makes future platforms more dynamic in their processing power. DF does not deal naturally within these constraints. If we want to use DF with high dynamics of system configuration, we must improve on the Model of Computation.
We want to improve the MoC but we do not want to lose the good properties:

- Concept should be simple to understand
- It should be hierarchical
- It should provide CSDF equivalency or assurance of the same kinds of properties
As long as internal paths are loosely stateless, we can insure CSDF equivalency.
As long as internal paths are loosely stateless, we can insure CSDF equivalency.

We can modelize dynamic data-paths with these.
A first example: best-of several paths/dynamic paths
Case of edge detections
Edge detection
Edge detection
Edge detection
Another example: redundancy and speculation

![Diagram showing redundancy and speculation](image-url)
Lots of current works with DataFlow paradigms aim at providing dynamic behavior and runtime versatility.
Related Works

Lots of current works with DataFlow paradigms aim at providing dynamic behavior and runtime versatility. They often fail on the ground of determinism.
Related Works

Lots of current works with DataFlow paradigms aim at providing dynamic behavior and runtime versatility. They often fail on the ground of determinism.

2 main frameworks aims at the same goals:

- **Scenario Aware DataFlows**: Filters can change their behavior according to several pre-defined scenarios.
- **Time-Triggered CSDF**: Augment the execution model of CSDF with time windows, and so is real-time aware.
Conclusion and Future work

- CSDF-equivalent model with accounting for dynamic configurations
- Can answer to a good amount of emerging issues
- It is time constraints compatible as long as filters have good timing properties (as in real-time systems)
Conclusion and Future work

- CSDF-equivalent model with accounting for dynamic configurations
- Can answer to a good amount of emerging issues
- It is time constraints compatible as long as filters have good timing properties (as in real-time systems)

- there is still some open questions about how a compiler can account for communication delays in a automatic way (without having the programmers to overspecify themselves)
Thanks!