Requirements for Application Software and Hardware imposed by Temporal Analysis Techniques

Marco Bekooij (NXP Semiconductors) and
Maarten Wiggers (University of Twente)

Outline

- Context
  - Introduced by Marco Bekooij
  - Focus: computing settings that guarantee temporal behaviour
- Application requirements
  - Definition of classes of applications
  - Guarantees on temporal behaviour only possible for a class of applications
- Architecture requirements
  - Different classes of run-time arbiters
  - Model independent of other jobs only possible for a class of schedulers
- Experimental results
- Conclusion

Context

- Applications
  - Jobs process streams of data
  - Job is a task graph
  - Multiple jobs executing concurrently
- Architecture
  - Multi-processor
  - Local memories and caches + SDRAM
- Real-time
  - Firm real-time constraints (deadline miss causes significant drop in quality)
  - Worst case execution times of tasks potentially unsafe (e.g. caches)
  - Data-dependent execution rates

Focus

- Objective
  - Compute settings, e.g. scheduler settings and buffer capacities, that **guarantee** lower bound on throughput and upper bound on latency of a **job**

- Guarantees on temporal behaviour requires
  - **Functionally deterministic** jobs
  - Guarantees on **deadlock-freedom**

- Guarantees on temporal behaviour of a job requires
  - Run-time schedulers that guarantee **resource budgets**

Focus

- Objective
  - Compute settings, e.g. scheduler settings and buffer capacities, that **guarantee** lower bound on throughput and upper bound on latency of a **job**

- Guarantees on temporal behaviour requires
  - **Functionally deterministic** jobs
  - Guarantees on **deadlock-freedom**

- Guarantees on temporal behaviour of a job requires
  - Run-time schedulers that guarantee **resource budgets**

guarantees ⇒ restrictions

Deadlock-freedom

- Guarantees on throughput and latency requires
  - guarantees on progress, i.e. guarantees on deadlock-freedom
- For **Turing complete** models, deadlock-freedom **undecidable**
  - Otherwise halting problem decidable
- Execution in bounded memory is necessary for deadlock-freedom
  - For some dataflow models “execution in bounded memory” is **decidable**
  - These dataflow models are **not Turing complete**

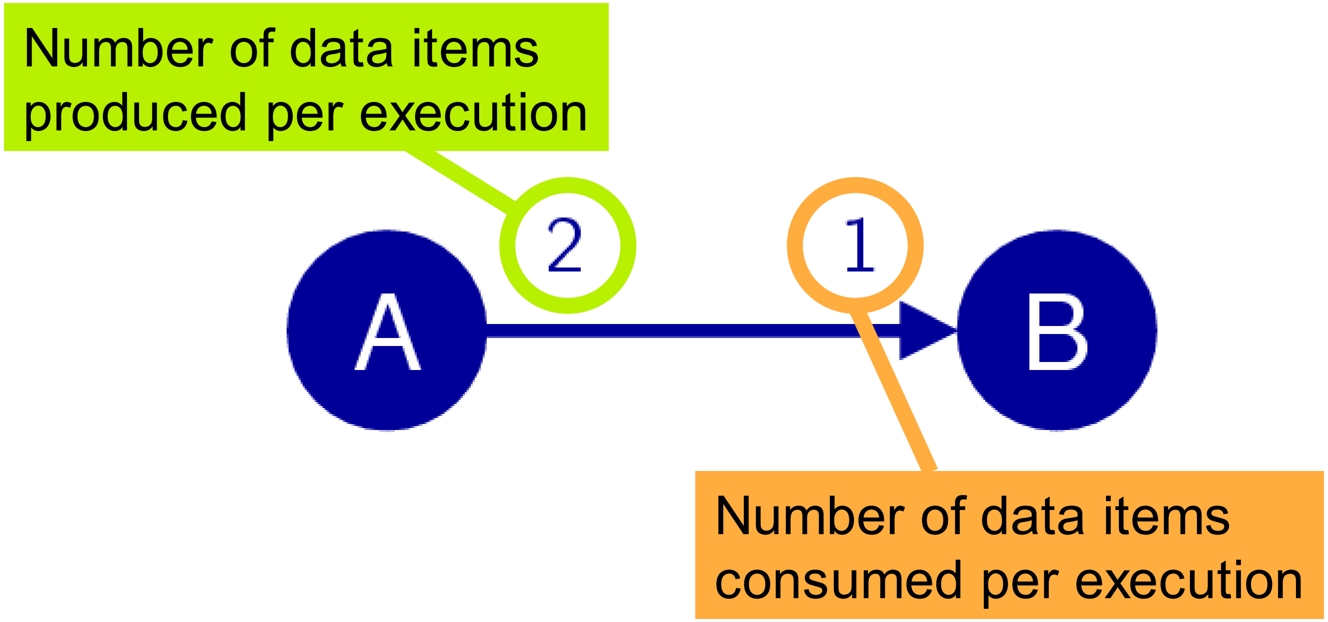
Consistency

- **Consistency** is necessary for execution in **bounded memory**
- Transfer quanta on edges determine relative execution rates

Synchronous Dataflow. Lee and Messcherschmitt. 1987
Consistency in Dataflow Graphs. Lee. 1991

Consistency

- **Consistency** is necessary for execution in **bounded memory**
- Transfer quanta on edges determine relative execution rates



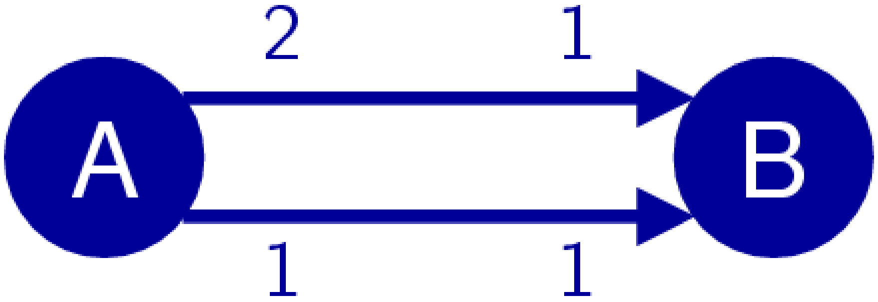
Number of data items produced per execution

Number of data items consumed per execution

Consistency

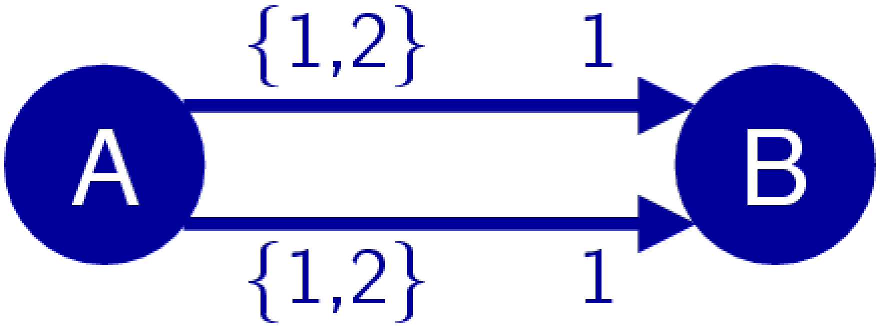
- Transfer quanta on edges determine relative execution rates

- Multiple paths between two actors
  - Requires **check** whether their exist execution rates with bounded memory



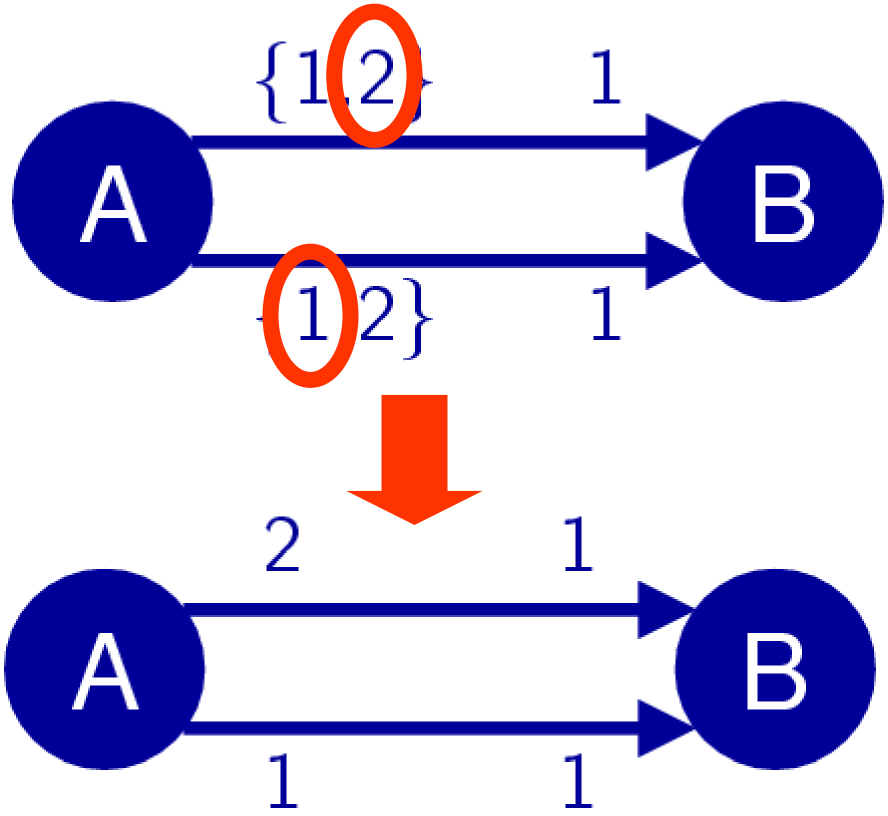
Consistency

- Fixed transfer quanta cannot model data-dependent behaviour
- Specification of **intervals** is **insufficient**



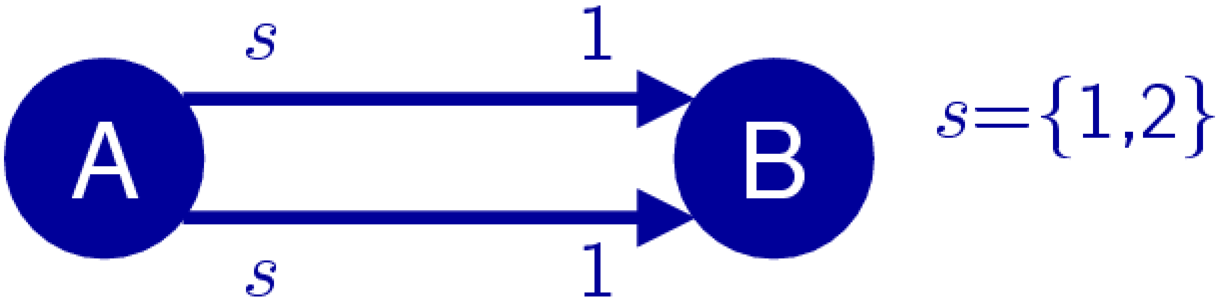
Consistency

- Specification of intervals is insufficient



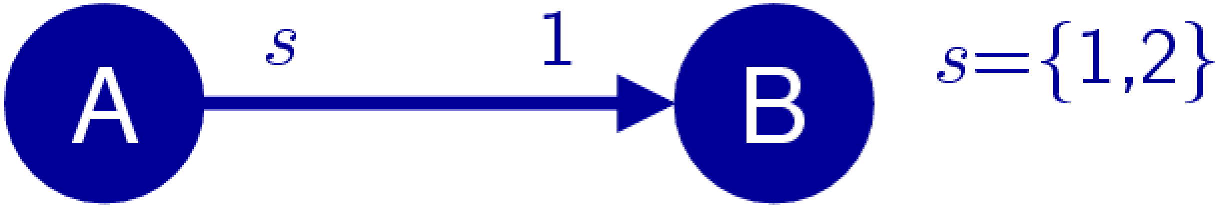
Consistency

- Fixed transfer quanta cannot model data-dependent behaviour
- Specification of intervals is insufficient
- Therefore introduce transfer **parameters** to create coupling



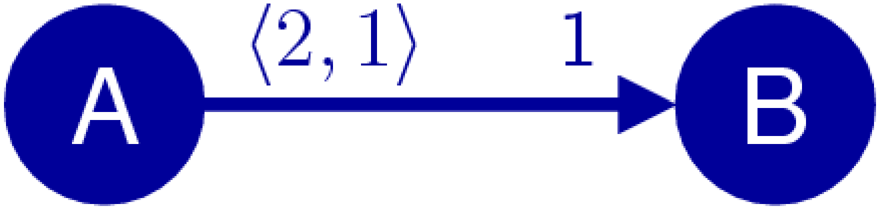
Scheduling

- For this (Variable-Rate Dataflow) graph no periodic schedule exists
- Number of executions of A relative to B varies with value of ***s***
  - Value of ***s*** can depend on the stream of data being processed



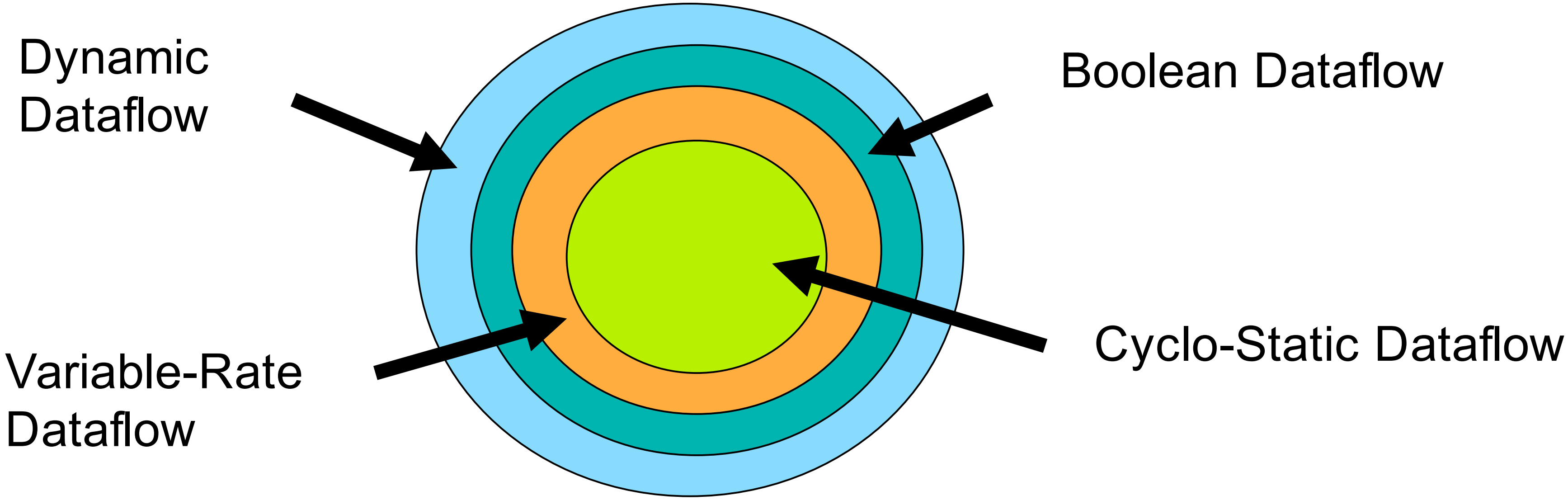
Scheduling

- For this (Cyclo-Static Dataflow) graph a **static-order schedule** exists
- For instance AABBB
- Static-order scheduling is **efficient**
  - No scheduling overhead
  - No intra-processor synchronisation costs



Classification

- Set of all applications (incl. Dynamic Dataflow)
  - Set of all functionally deterministic applications (incl. BDF)
    - Set of all provably deadlock-free applications (incl. VRDF)
      - Set of applications with static-order schedule (incl. CSDF)

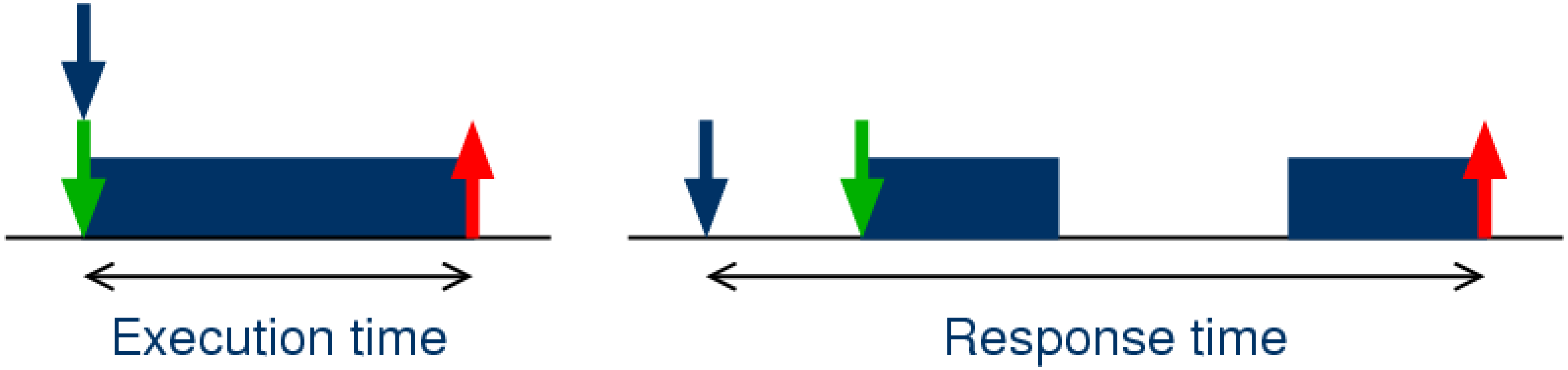


Message : application requirements

- Different classes can be identified
- Differentiation necessary to guarantee temporal behaviour
  - Functional determinism
  - Deadlock-freedom
- Differentiation necessary for cost-efficient scheduling
  - Static-order scheduling
- Main research challenge to define models
  - For which deadlock-freedom is decidable
  - Data-dependent synchronisation behaviour ⇒ no static-order scheduling
  - E.g. variable-rate dataflow (Wiggers, RTAS’08)

Response times

- Enabling time : sufficient data and space is available in buffers
- **Execution time** of code-segment == time between enabling and finish
  - Execution in isolation
  - Enabling time == start time
- **Response time** of code-segment == time between enabling and finish
  - Resource is shared
  - Enabling time ≠ start time



Run-time scheduling

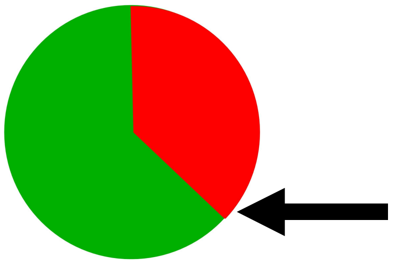
- Response time depends on
  - Execution time
  - Interference from other tasks
- Interference can depend on
  - Number of activations of other tasks
  - Execution times of other tasks
- Leads to three types of schedulers
  1. RT depends on activations & execution times
  2. RT depends on execution times
  3. RT independent

Run-time scheduling (cont.)

- Dependence on: activations & execution times
  - Classic single-processor real-time schedulers
  - E.g. static priority pre-emptive
- Dependence on: execution times
  - Latency-rate servers
  - E.g. round-robin
- Independent: interference bounded by construction
  - Budget schedulers
  - E.g. time-division multiplex

Response time calculation

- Time-division multiplex (TDM) is a budget scheduler
- Classical response time computation
  - Independent of arrival times
  - Assumes worst-case enabling time

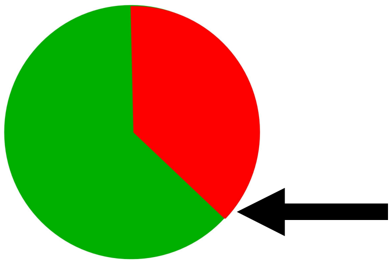


Worst-case enabling time (TDM)

Response time calculation

- Time-division multiplex (TDM) is a budget scheduler
- Classical response time computation
  - Independent of arrival times
  - Assumes worst-case enabling time

$$wcrt = wcet + (P - B)\left\lceil \frac{wcet}{B} \right\rceil$$

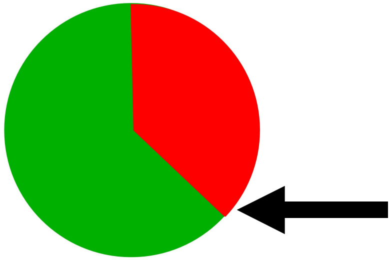


Worst-case enabling time (TDM)

Response time calculation

- Time-division multiplex (TDM) is a budget scheduler
- Classical response time computation
  - Independent of arrival times
  - Assumes worst-case enabling time

$$wcrt = wcet + (P - B)\left\lceil \frac{wcet}{B} \right\rceil$$



Worst-case enabling time (TDM)

- Upper bound on finish time
  - Independent of previous finish time

$$f(i) = e(i) + wcet + (P - B)\left\lceil \frac{wcet}{B} \right\rceil$$

Improved response time calculation

- Traditional model
  - Does not capture multiple consecutive executions in one slice
  - Correct from a latency point of view
  - Too pessimistic from a throughput point of view
- If you know that enabling time is before previous finish time, then you do not always need to assume the initial pre-emption

Latency-Rate servers. Stiliadis and Varma. 1998

Improved response time calculation

- Traditional model
  - Does not capture multiple consecutive executions in one slice
  - Correct from a latency point of view
  - Too pessimistic from a throughput point of view
- If you know that enabling time is before previous finish time, then you do not always need to assume the initial pre-emption

$$f(i) = \max(e(i) + P - B, f(i-1)) + P\frac{x(i)}{B}$$

Improved response time calculation

- Traditional model
  - Does not capture multiple consecutive executions in one slice
  - Correct from a latency point of view
  - Too pessimistic from a throughput point of view
- If you know that enabling time is before previous finish time, then you do not always need to assume the initial pre-emption

$$f(i) = \max(e(i) + P - B, f(i-1)) + P\frac{x(i)}{B}$$

Worst-case enabling time

+

initial pre-emption

Improved response time calculation

- Traditional model
  - Does not capture multiple consecutive executions in one slice
  - Correct from a latency point of view
  - Too pessimistic from a throughput point of view
- If you know that enabling time is before previous finish time, then you do not always need to assume the initial pre-emption

$$f(i) = \max(\underbrace{e(i) + P - B}_{\text{Worst-case enabling time } + \text{ initial pre-emption}}, \underbrace{f(i-1)}_{\text{Previous finish}}) + P\frac{x(i)}{B}$$

Improved response time calculation

- Traditional model
  - Does not capture multiple consecutive executions in one slice
  - Correct from a latency point of view
  - Too pessimistic from a throughput point of view
- If you know that enabling time is before previous finish time, then you do not always need to assume the initial pre-emption

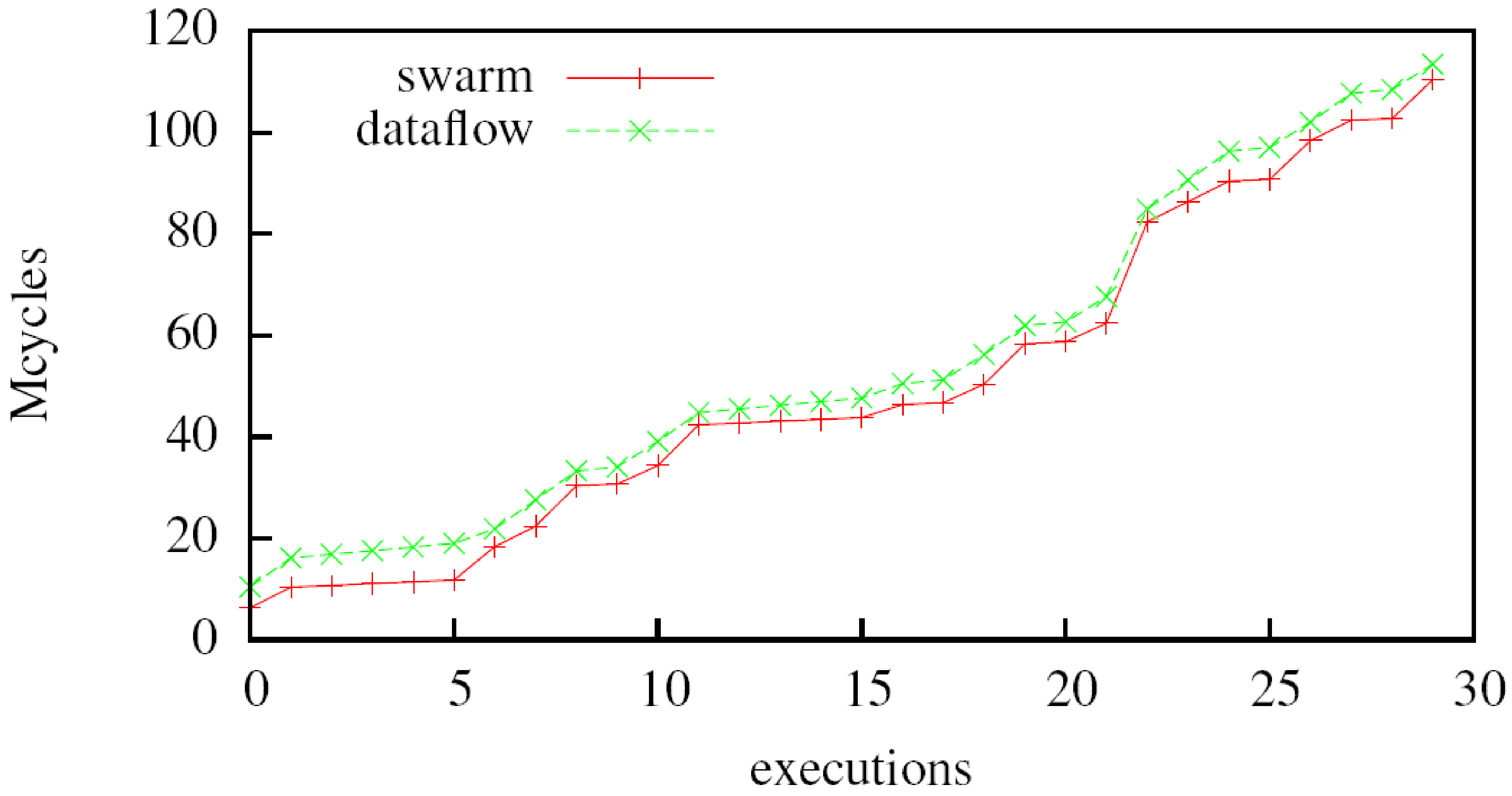
$$f(i) = \max(\underbrace{e(i) + P - B}_{\text{Worst-case enabling time + initial pre-emption}}, \underbrace{f(i-1)}_{\text{Previous finish}}) + \underbrace{P\frac{x(i)}{B}}_{\text{Execution on a P/B times slower processor}}$$

Experiment

- Aim: test accuracy of dataflow model
- Set-up
  - Producer-consumer with variable production quantum
  - 2 ARM processors that share one double ported memory
  - Cycle-accurate systemC model (using SWARM)
  - Execution time producer > time slice producer
  - Execution time consumer << time slice consumer

$s = [0,5]$

Experimental results



Message: architectural requirements

- Different classes of schedulers can be identified
- Conservative model independent of other jobs only possible for budget schedulers
- Showed a tight conservative model for time-division multiplex
  - Current (submitted) work shows extension to other budget schedulers

Conclusion

- Objective
  - Compute settings, e.g. scheduler settings and buffer capacities, that **guarantee** lower bound on throughput and upper bound on latency of a **job**
- Guarantees on temporal behaviour requires
  - **Functionally deterministic** jobs
  - Guarantees on **deadlock-freedom**
- Guarantees on temporal behaviour of a job requires
  - Run-time schedulers that guarantee **resource budgets**

guarantees ⇒ restrictions

Questions?