Job-Level Batching for Software-Defined Radio on Multi-core

Abby Eisenklam
University of Pennsylvania*

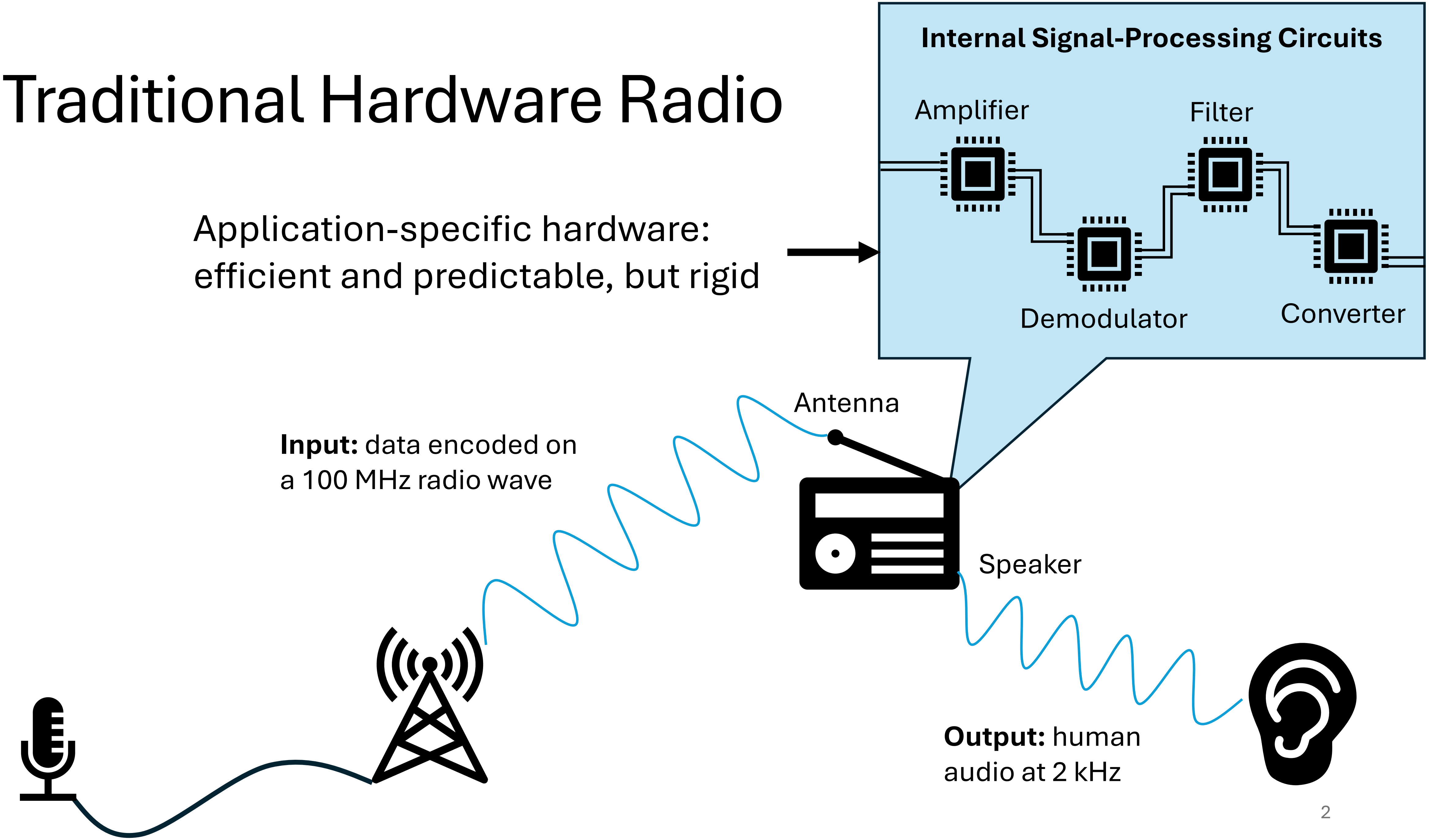
Will Hedgecock
Vanderbilt University

Bryan C. Ward
Vanderbilt University

*work done while at Vanderbilt

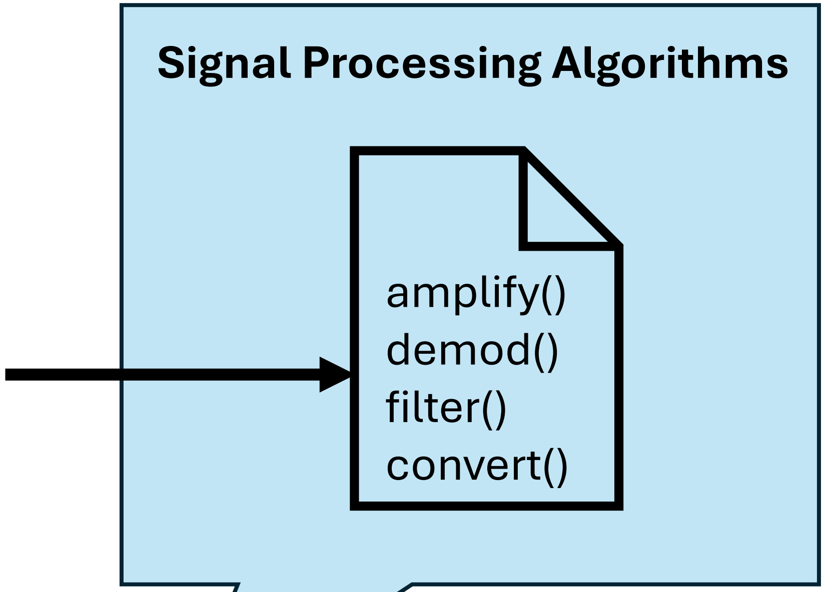
Traditional Hardware Radio

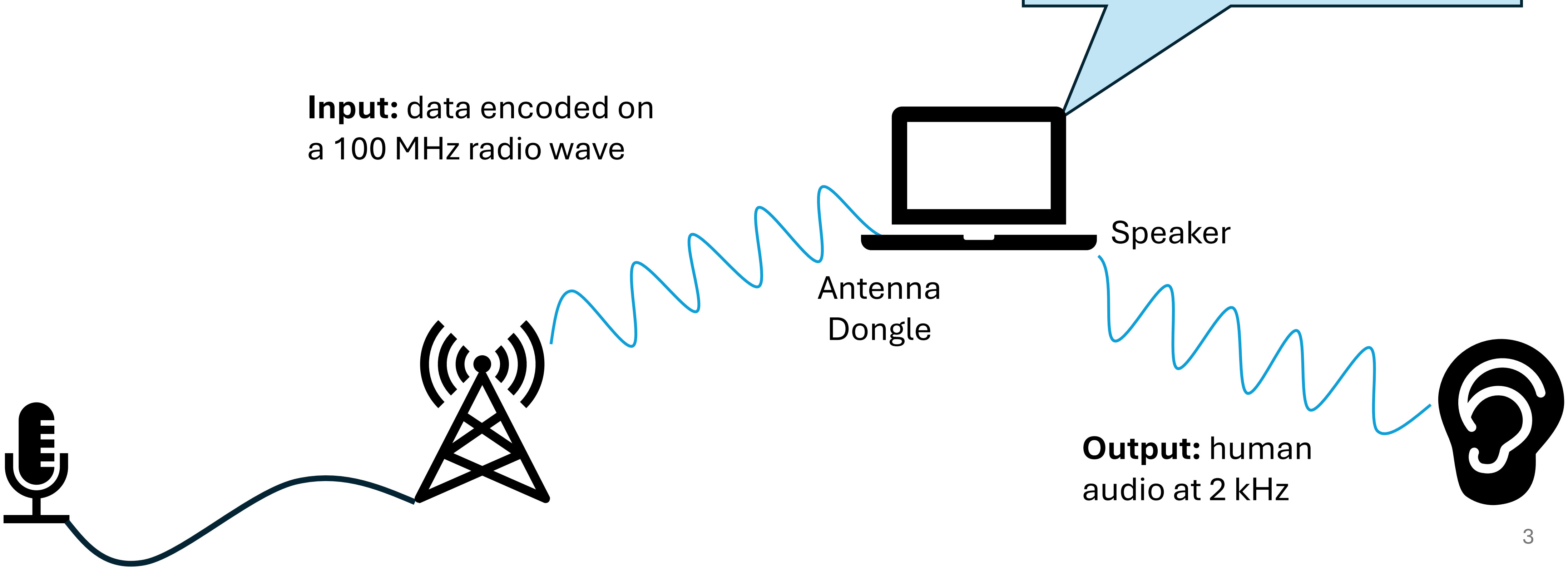
Application-specific hardware:
efficient and predictable, but rigid



Software-Defined Radio

Software applications on GPPs:
flexible, upgradeable, and adaptable





Why SDR on General Purpose Processors?

Flexibility	• Support multiple frequency bands and communication protocols • More complex algorithms • Spectrum sensing, signal classification, etc.
Upgradeability	• Efficient and cost-effective updates when technology/protocols change • No need to buy new hardware, reconfigure FPGA, etc.

Adaptability	• Detect and adapt in real-time to congestion, signal interference, or malicious jamming detected in environment*
	    • Dynamic spectrum sensing
	    • Cognitive (autonomous) radio

*DARPA. DARPA eyes adaptive, real-time processors for future AI-enabled radios. https://www.darpa.mil/news-events/2022-10-06.

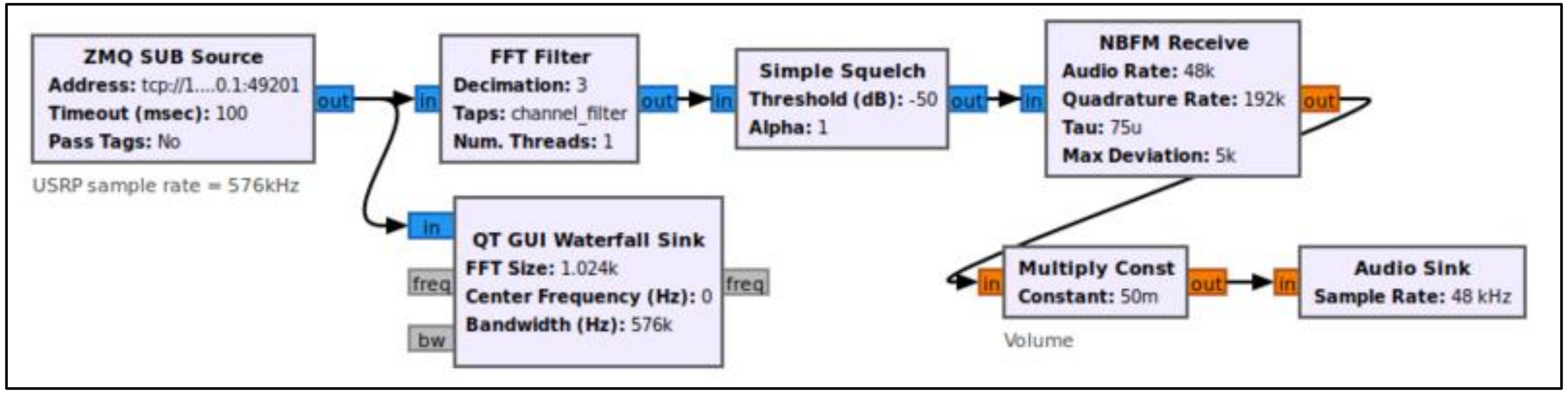
Application Domains for SDR

- Wireless networks (e.g. WiFi, Bluetooth, IoT)
- Cellular technologies (e.g. 5G)
- Satellite communications
- Spectrum sensing

Multi-core SDR is not yet widely used in these domains due to **unpredictable latency** and **resource inefficiency**.

GNU Radio

- Developers can build applications by combining signal-processing “blocks”
  - filters, amplifiers, etc.
- Blocks are connected by FIFO buffers



Narrowband FM Receiver in GNU Radio

GNU Radio “Scheduler”

- Each block is assigned its own thread
- Threads are woken up when their input buffers are at least half full
  - PRO: Reduces context switching, prioritizes throughput
  - CON: Nondeterministic input size, variable execution times
- Actual scheduling decisions left to underlying OS

GNU Radio provides no control over factors that affect real-time performance*

*B. Bloessl, M. M üller, and M. Hollik. Benchmarking and profiling the GNU radio scheduler. In Proceedings of the 9th GNU Radio Conference ’19.

What can we do?

- Goal: make SDR applications more predictable and hardware-efficient on general-purpose multiprocessors

- Challenge: traditional real-time scheduling models target applications with slower sampling rates
  - Period of 10ms corresponds to frequency of 10 Hz
  - Input signal frequency can be on the order of 10 GHz, 10^9 times faster
- Context switch overhead is large compared to computational cost
- Batch processing samples is fundamental to supporting SDR on multi-core

Research Questions

1. What is the relationship between batch size and execution time?

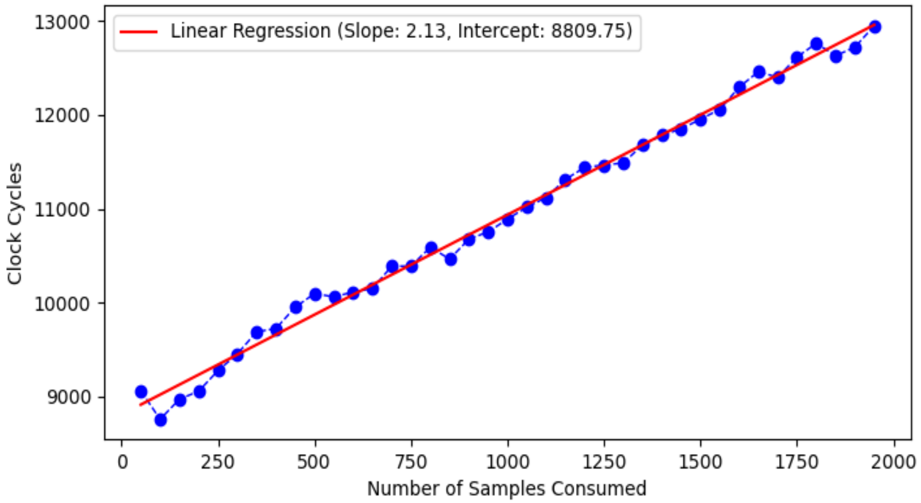
2. How do we choose batch sizes such that an application's latency requirements and resource constraints are met?

Outline

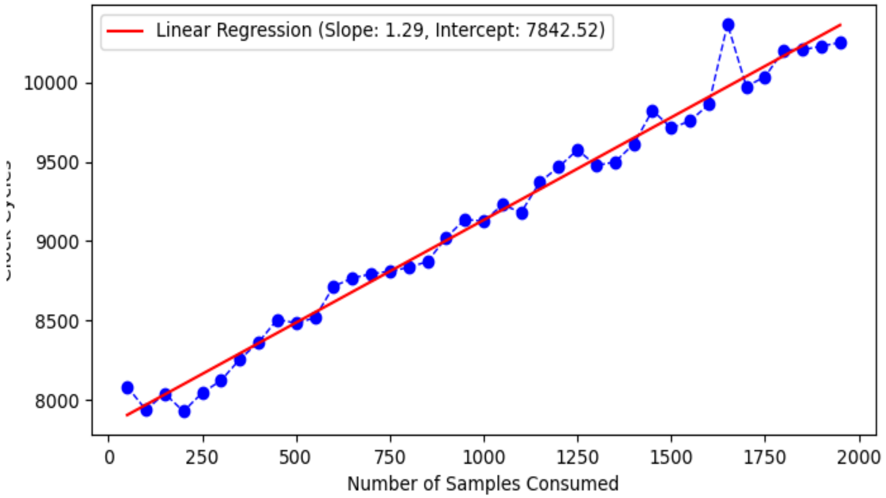
1. Motivation
2. GNU Radio Experiments
3. The Marginal Cost Model
4. Batching Techniques
5. Latency Analysis
6. Evaluation
7. GNU Radio Case Study

Batch Size vs. Execution Time in GNU Radio

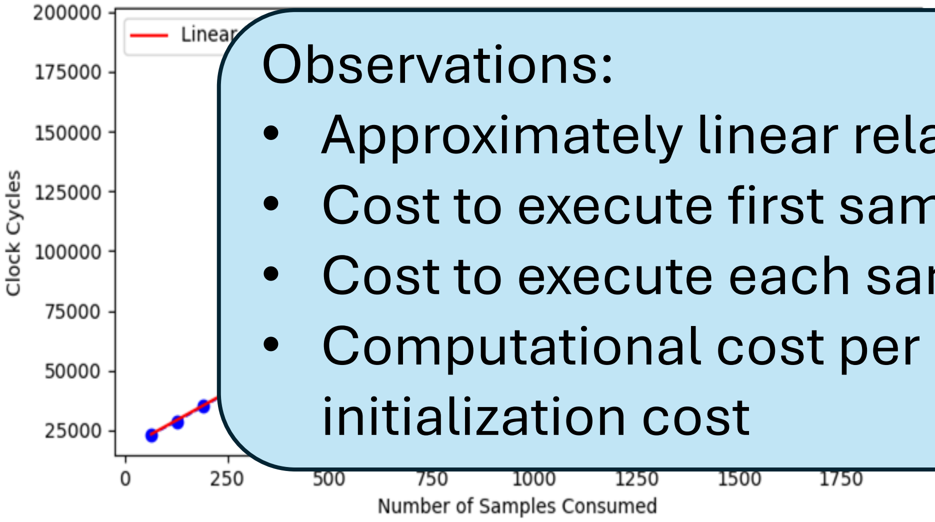
Frequency Shift



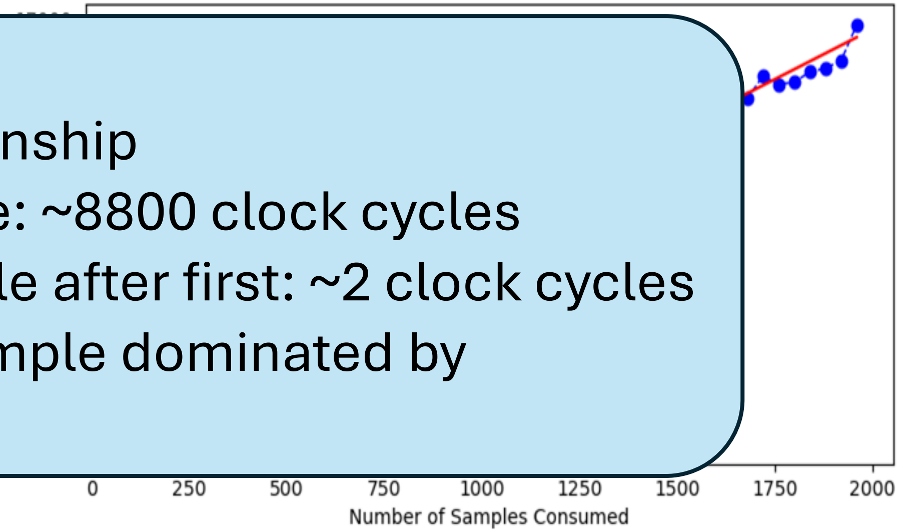
Amplify



FIR Filter



Constant Add



Observations:

- Approximately linear relationship
- Cost to execute first sample: ~8800 clock cycles
- Cost to execute each sample after first: ~2 clock cycles
- Computational cost per sample dominated by initialization cost

Outline

1. Motivation
2. GNU Radio Experiments
3. The Marginal Cost Model
4. Batching Techniques
5. Latency Analysis
6. Evaluation
7. GNU Radio Case Study

The Marginal Cost Model

- The relationship between batch size (*N*) and execution time (*e*) is linear
- Y-intercept represents a fixed initialization cost (*I*)
- Slope represents the marginal cost (Δ) to process one additional sample
- The execution time of a task τ is given by:

$$e_\tau = I_\tau + N \cdot \Delta_\tau$$

Batching Jobs with the Marginal Cost Model

- GNU Radio experiments showed that the initialization cost, I >>> marginal cost, Δ
  - by a factor of 100x to over 1000x
- We can batch jobs to reduce the number of initialization costs needed to do the same amount of work

Execution time before batching

Execution time after batching

Research Questions

1. What is the relationship between batch size and execution time?

2. How do we choose batch sizes such that an application's latency requirements and resource constraints are met?

   To answer this question, we need a real-time framework.

Model and System Requirements

Model of Computation

- DAG-based
- Dataflow
- Fixed source frequency
- Blocks have some consume-to-produce ratio

System Requirements

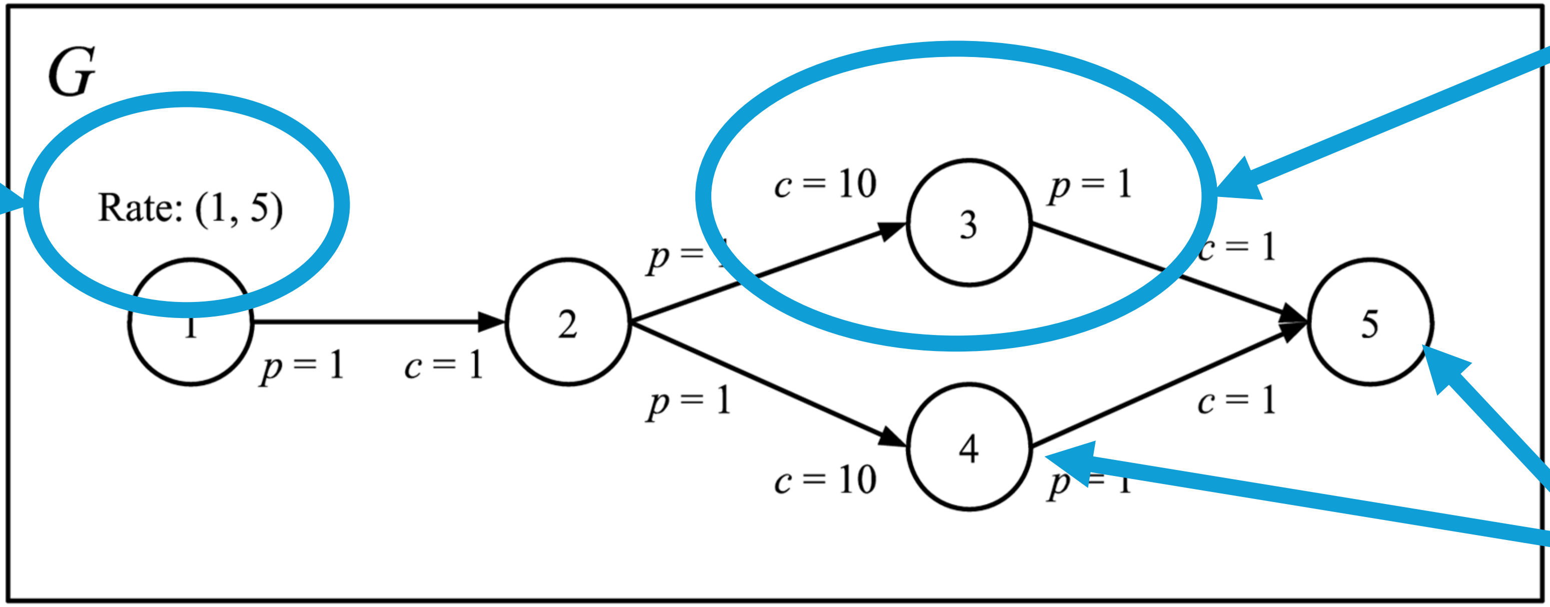
- Sufficient cores for application frequency/throughput
- No strict deadlines, but bounded latency

Even better:

- Efficiently utilized cores
- End-to-end latency bounds

Processing Graph Method (PGM)

Source node executes once every 5 time units



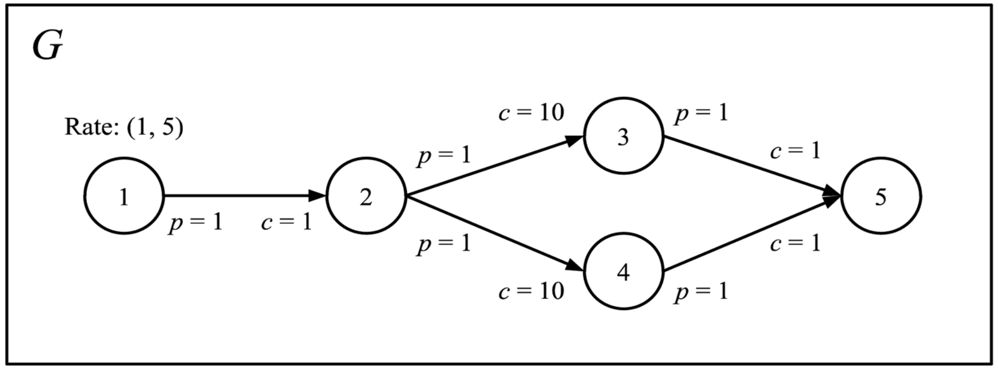
Node 3 needs to consume 10 data samples to produce one output

Rates of non-source nodes calculated using source rate and produce-consume ratios*

*S. Goddard. On the management of latency in the synthesis of real-time signal processing systems from processing graphs. PhD thesis, UNC, Chapel Hill, NC, 1998.

PGM Graphs on Multi-core

- PGM graphs can be soft real-time scheduled on multi-core with no utilization loss*
  - Calculate rates for each node of a PGM graph → transform to independent sporadic tasks
- Tardiness w.r.t. individual deadlines is bounded under G-EDF**



$\tau_1 = (5, 5, WCET_1)$

$\tau_2 = (5, 5, WCET_2)$

$\tau_3 = (50, 50, WCET_3)$....

Replace with marginal cost model

*C. Liu and J. Anderson. Supporting soft real-time DAG-based systems on multiprocessors with no utilization loss. In RTSS '10.

**U. Devi, Soft Real-Time Scheduling on Multiprocessors. PhD thesis, Department of Computer Science, UNC, NC, 2006.

Outline

1. Motivation
2. GNU Radio Experiments
3. The Marginal Cost Model
4. Batching Techniques
5. Latency Analysis
6. Evaluation
7. GNU Radio Case Study

Uniform Batching

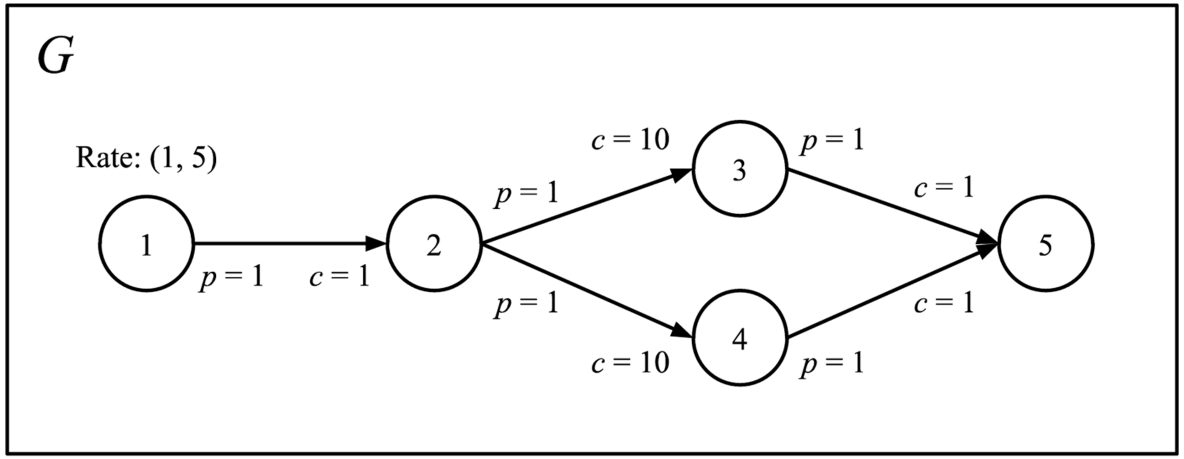
- Suppose we have a PGM graph G with batch size N
- Then the sporadic parameters for each $\tau \in G$ are given by:

$$\tau = (Np_\tau, \mathrm{N}d_{\tau,}\ l_\tau + N\Delta_\tau)$$

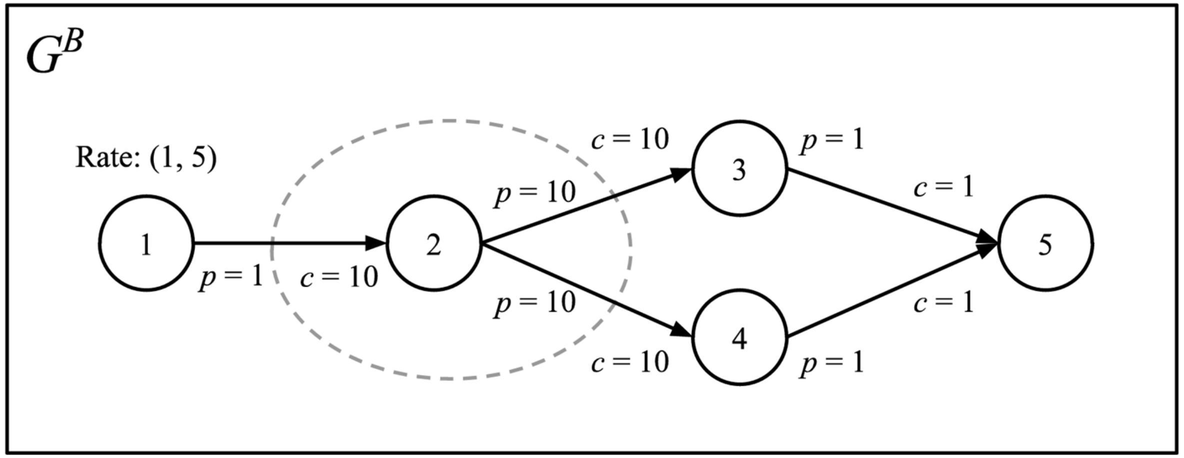
where p_τ and d_τ are calculated using source rate and unit input sizes.

Invocation of each τ is delayed until N data samples arrive, effectively batching N jobs of τ into a single job.

Rate-Exploiting Batching



Batch Node 2 by N = 10



Consequences:

- Rate of node 2 decreases by factor of 10
- Rates of successor nodes are unaffected
- Reduced utilization without impacting output rate

Outline

1. Motivation
2. GNU Radio Experiments
3. The Marginal Cost Model
4. Batching Techniques
5. Utilization and Latency Analysis
6. Evaluation
7. GNU Radio Case Study

Intuition for Utilization Decrease

Execution time before batching

Execution time after batching

Taking a conservative estimate, suppose $I = 100$, $\Delta = 1$, and $p = 300$. Then,

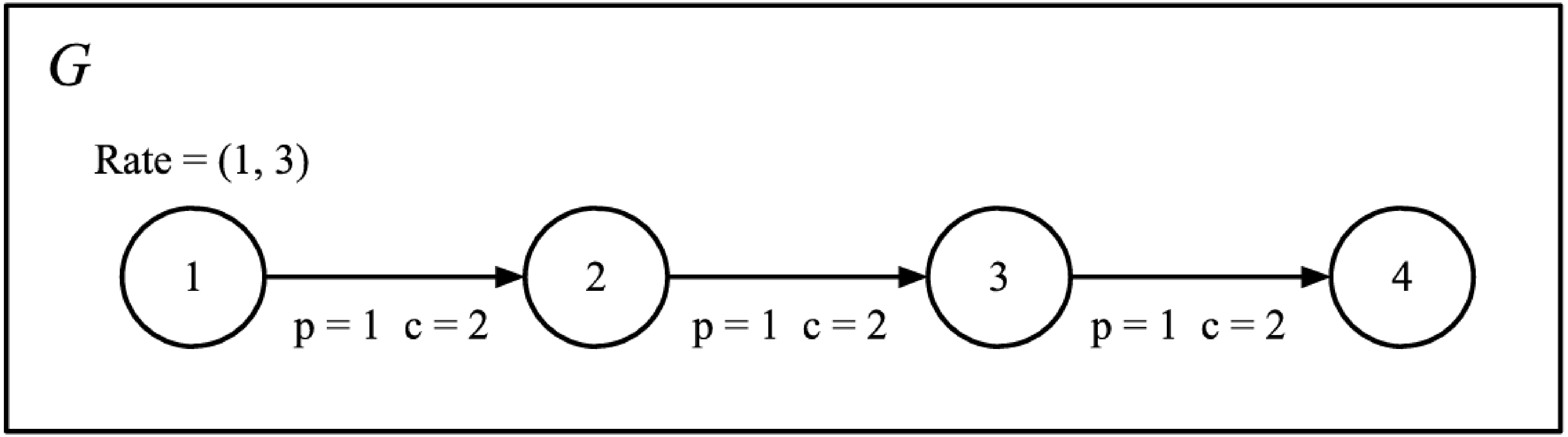
Utilization before batching:

- $U = (I + 2\,\Delta) / p$
  $= (100 + 2) / 300$
  $= 0.340$

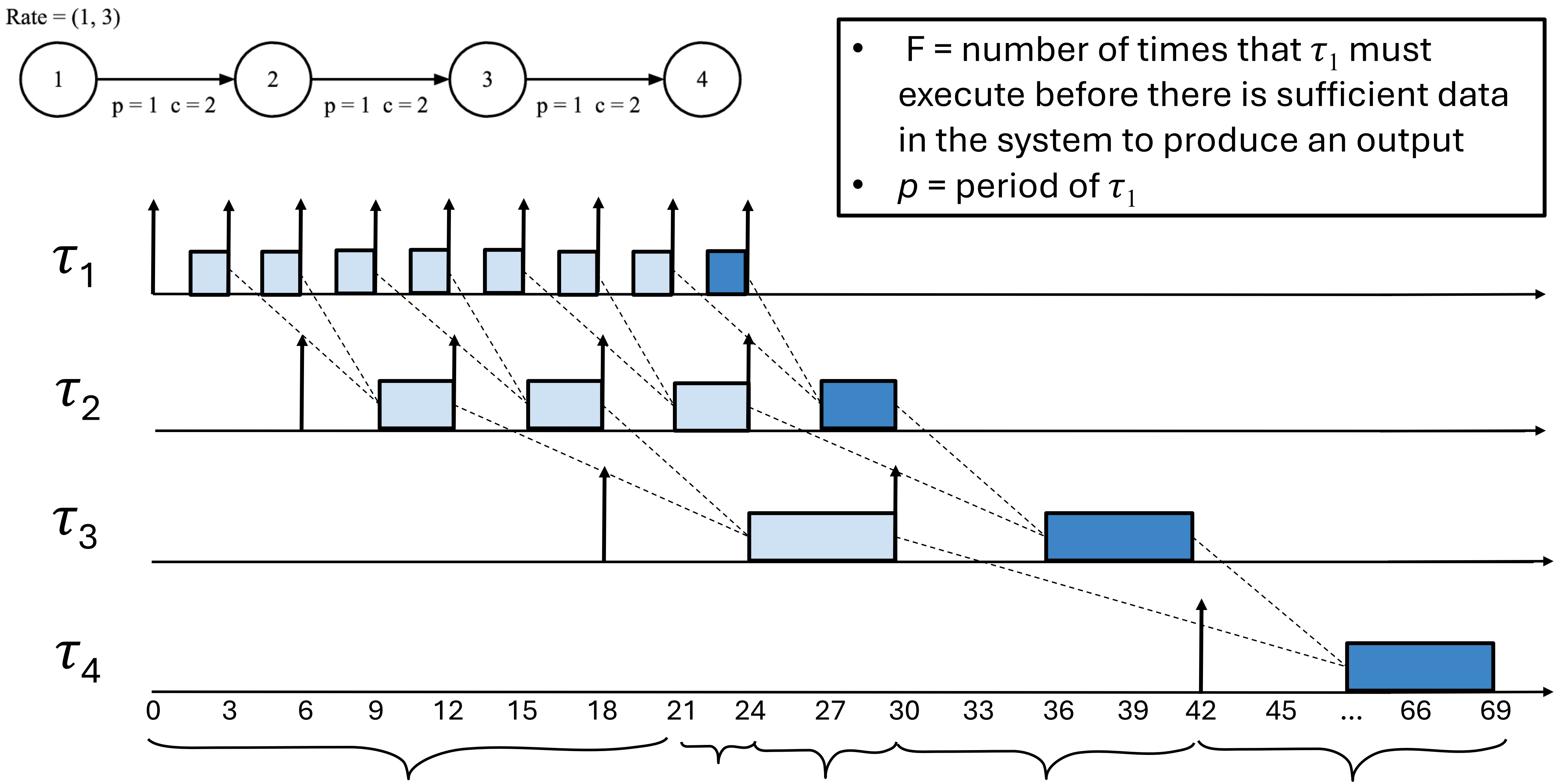
Utilization after batching:

- $U = (I + 6\,\Delta) / 3p$
  $= (100 + 6) / 900$
  $= 0.118$

Intuition for End-to-End Latency



- F = number of times that τ_1 must execute before there is sufficient data in the system to produce an output
- p = period of τ_1



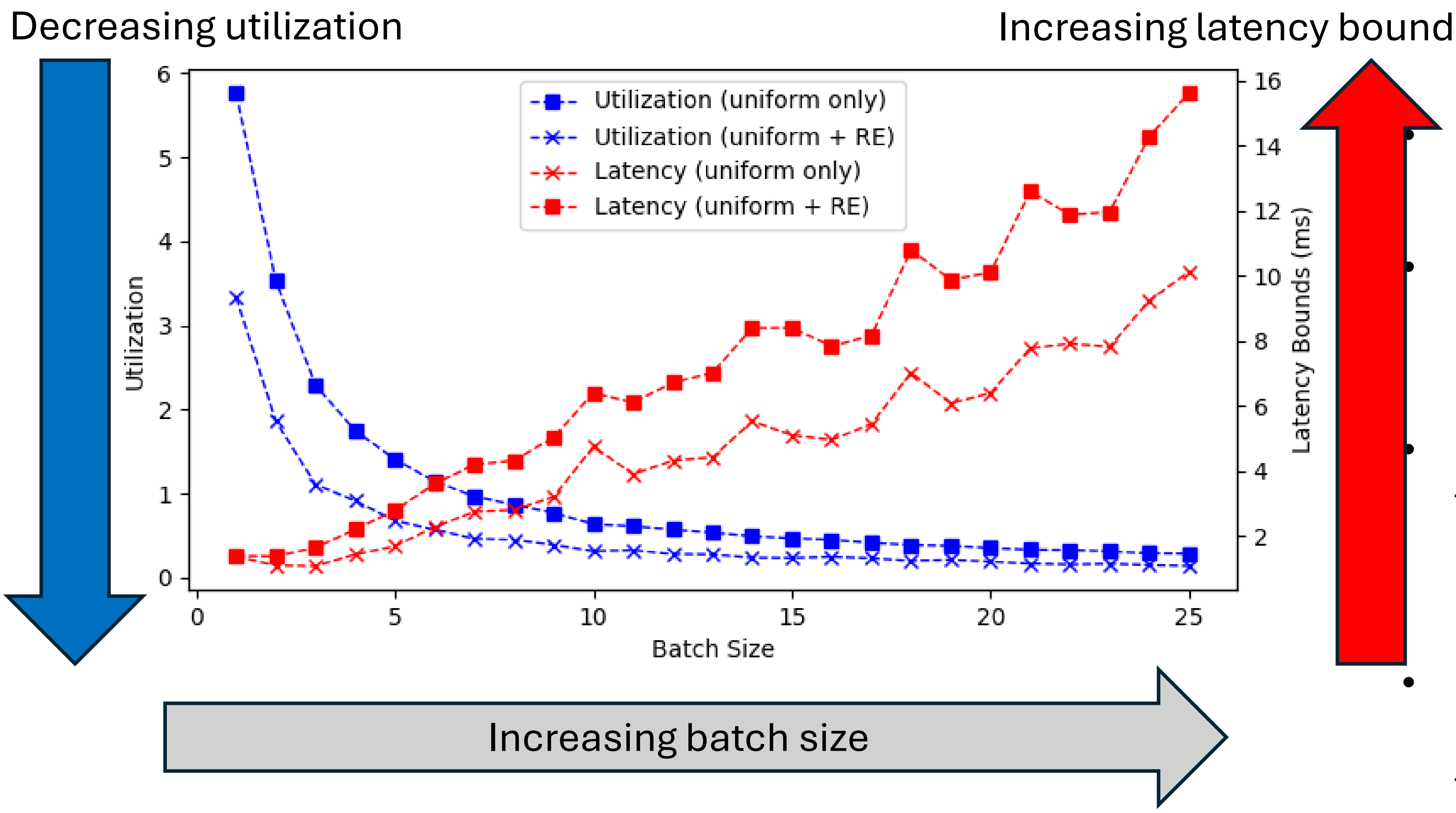
(F – 1)* p

Sum of worst-case response times along critical path (using bound for sporadic tasks under G-EDF)

Outline

1. Motivation
2. GNU Radio Experiments
3. The Marginal Cost Model
4. Batching Techniques
5. Latency Analysis
6. Evaluation
7. GNU Radio Case Study

Latency and Utilization vs. Batch Size

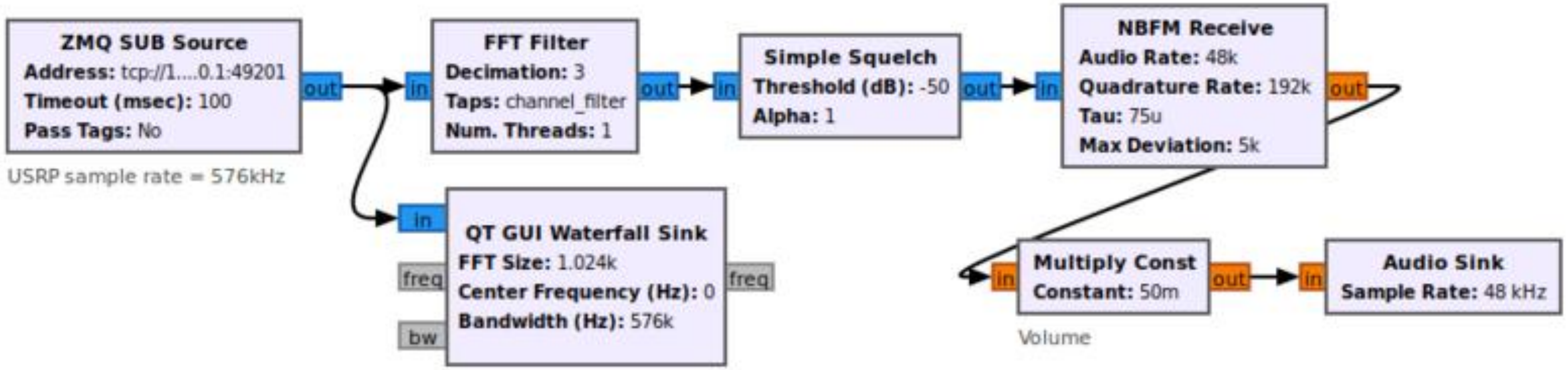


- Latency ~linear, diminishing returns in reducing utilization
- Rate-exploiting further reduces utilization with relatively small impact on latency
- Batching by N = 4 drops utilization from ~6 to ~2 with a 0.06ms latency penalty (can get to <1 core with additional rate-exploiting batching)
- System designers can use similar analysis to minimize utilization subject to a latency constraint

Each point is an average over 1000 synthetic PGMs, using execution times from GNU Radio experiments

Outline

Case Study: GNU Radio NBFM Receiver

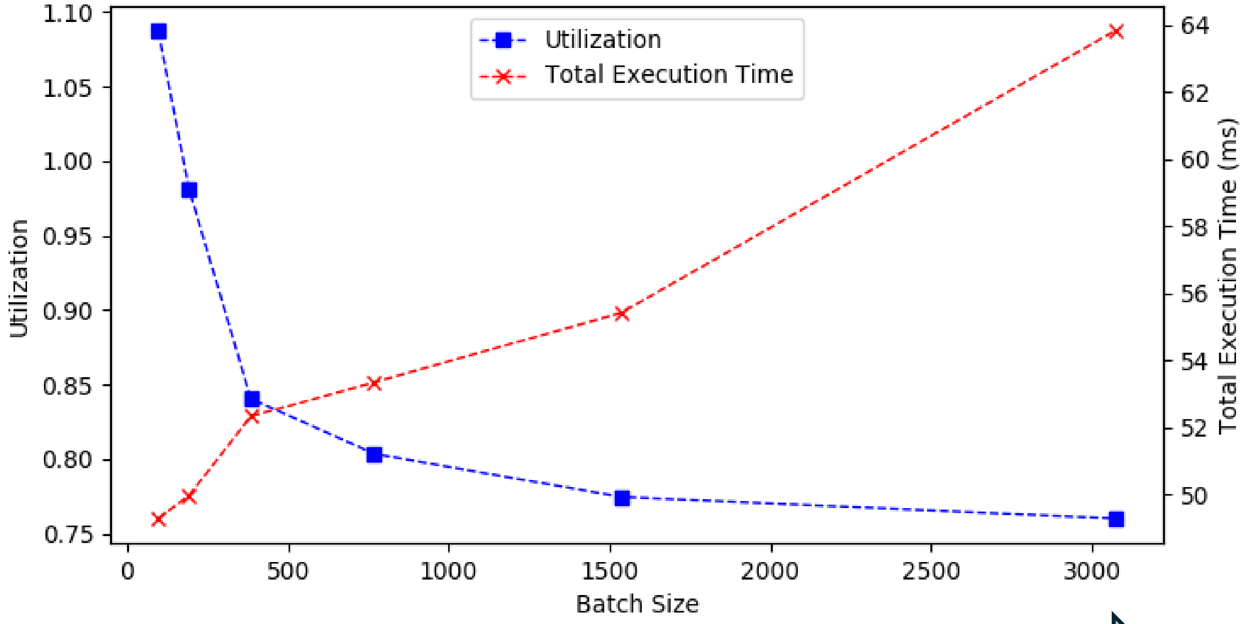


Experiment Setup

- Added an EDF API for GNU Radio that uses Linux SCHED_DEADLINE
- Isolated 3 out of 4 Raspberry Pi cores for GNU Radio tasks
  - Disabled kernel preemptions and IRQ requests from these cores
- Derived sporadic parameters for each block using source rate and produce-consume relationships
- Controlled for batch size using GNU Radio “Head” blocks and buffer sizes
- Measured max. latency using GNU Radio performance counters

Utilization and Latency vs. Batch Size for NBFM

Decreasing utilization



Increasing latency

Increasing batch size

The trends in latency and utilization as functions of batch size hold in a real SDR system.

Conclusion

- Validated the marginal cost model for batched sample processing in GNU Radio

- Presented uniform and rate-exploiting batching techniques

- Derived end-to-end latency bounds for multi-core PGM graphs scheduled under G-EDF as a function of batch size

- Evaluated randomly generated synthetic task systems to demonstrate the utilization-latency trade-off

- Extended GNU Radio to support Linux SCHED_DEADLINE and presented a case study applying the marginal cost model and batching to a real signal-processing application