Implementation-Aware Model Analysis: The Case of Buffer-Throughput Tradeoff in Streaming Applications

Kamyar Mirzazad Barijough, Matin Hashemi

Sharif University of Technology, Tehran, Iran

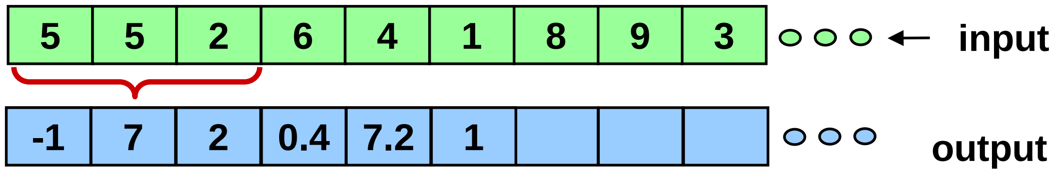
Volodymyr Khibin, Soheil Ghiasi

University of California, Davis, CA, USA

Streaming Applications

- Widespread
  - Cell phones, video conference, real-time encryption, graphics, HDTV editing, hyperspectral imaging, cellular base stations
- Properties
  - Infinite sequence of data items
  - At any given time, operates on a small window of this sequence
  - Fairly deterministic behavior
  - Throughput-Sensitive
- Implementation Platform
  - MPSoC is a competitive choice in the mix

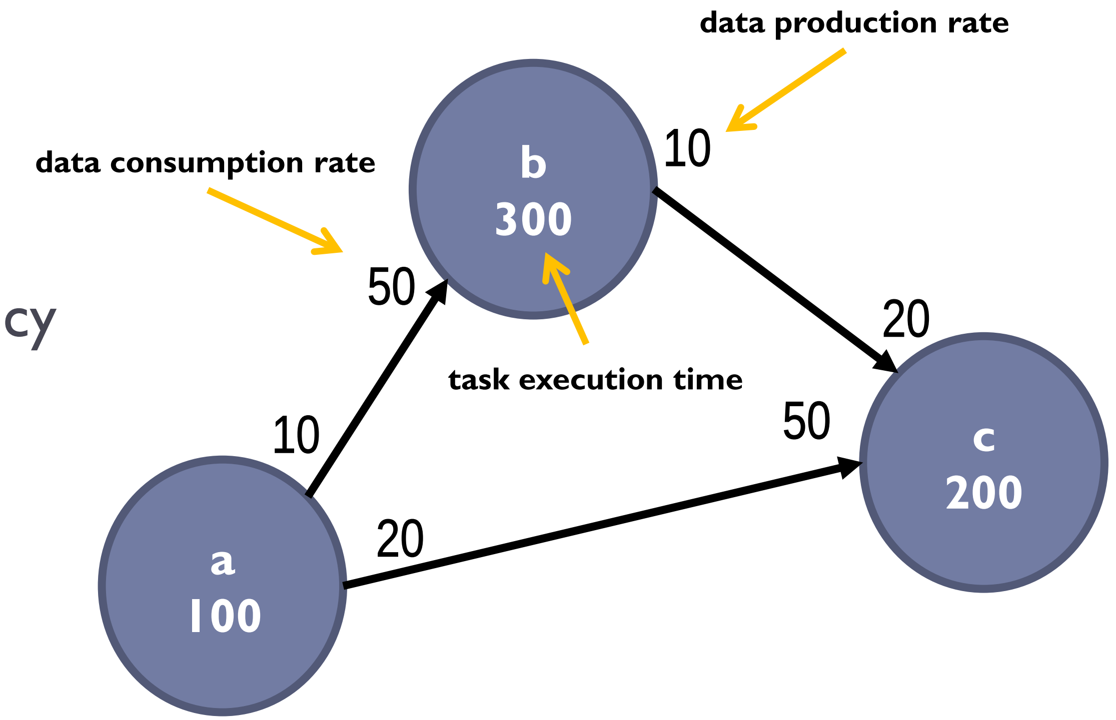
```
//53° around the z axis
const R[3][3]={
      {0.6,-0.8, 0.0},
      {0.8, 0.6, 0.0},
      {0.0, 0.0, 1.0}}
Rotation3D {
 for (i=0; i<3; i++)
  for (j=0; j<3; j++)
   B[i] += R[i][j] * A[j]
 }
```



Synchronous Dataflow (SDF) Model

- SDF model
  - a directed graph G(V,E)
  - Vertices represent actors
  - Edges represent inter-actor data dependency (FIFO communication semantics)
    - semantically have infinite storage capacity
  - Static data production and consumption rates

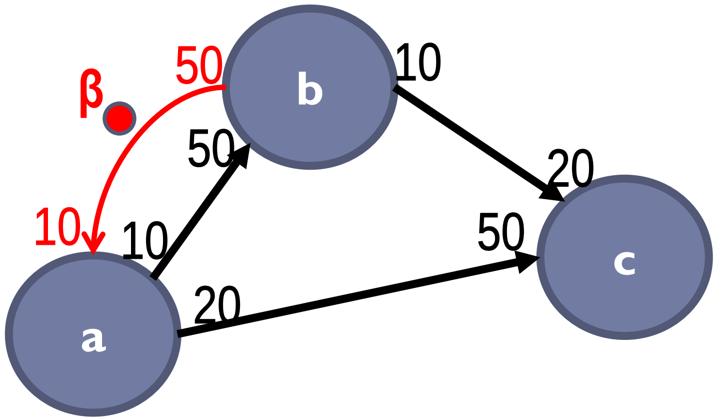
- Periodic static schedule



An example SDF graph

A Few Definitions

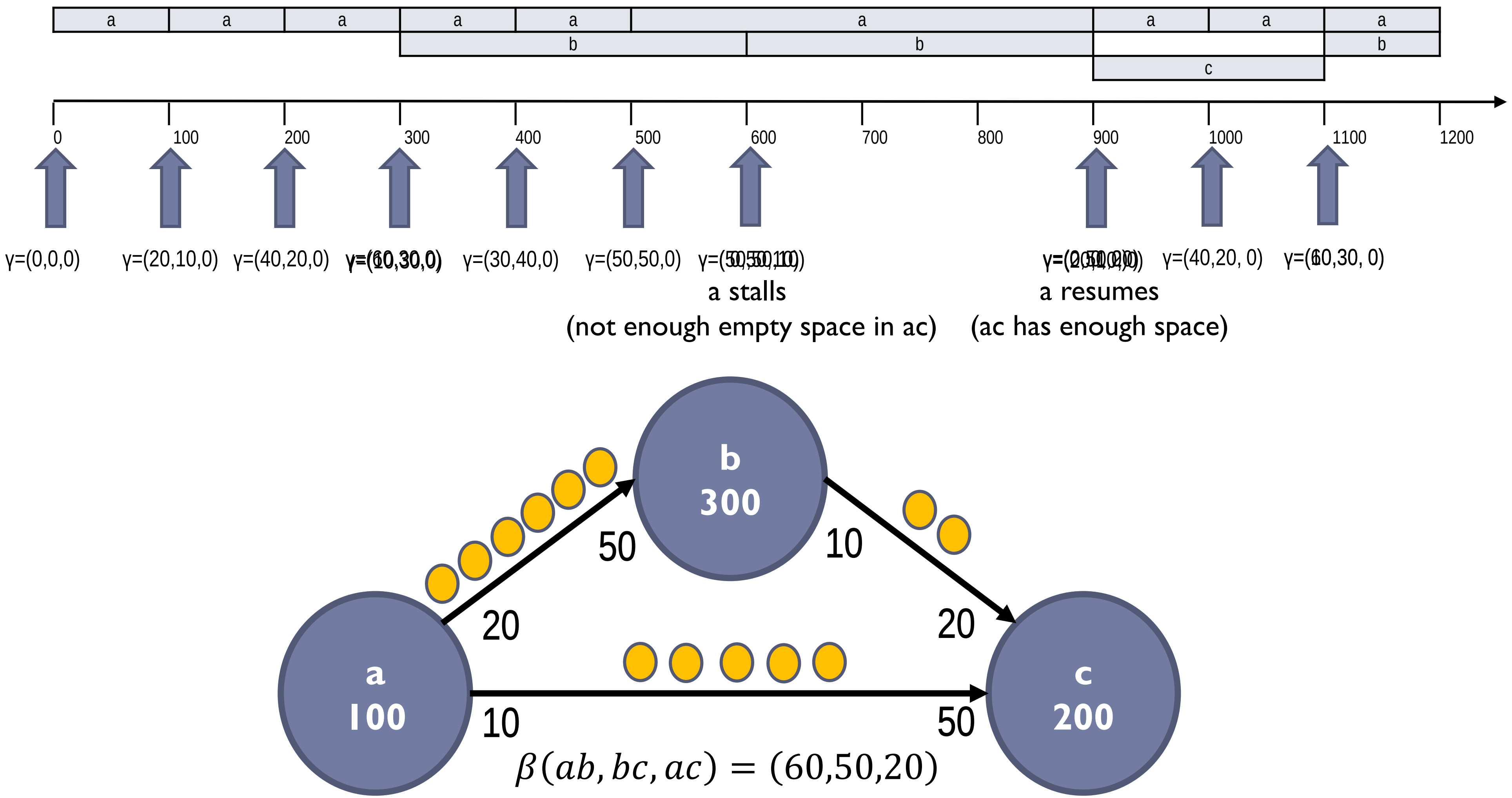
- Buffer size: storage capacity of inter-task channels
  - Infinite in the abstract model; in practice limited
  - Modeled as reverse channels with specific number of initial tokens & rates
- Throughput of an actor v: the average number of v firings per unit time
  - A number of factors, such as actor execution times, interprocessor buffer capacities and SDF graph cycles impact throughput.



Simplified SDF Operational Semantics

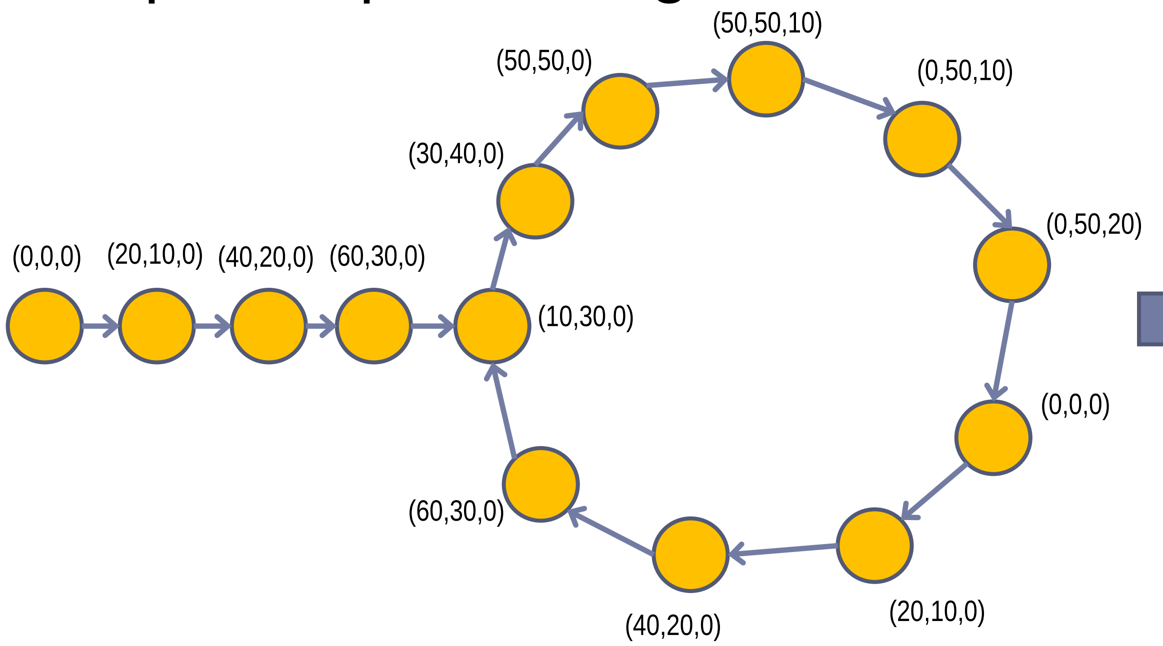
- An actor can fire only after sufficient number of input tokens are available on all of its input channels.
  - Otherwise firing is deferred
  - Upon firing all input tokens are consumed simultaneously
- After an actor completes its computation, sufficient space is required on all of its output channels to write the tokens produced.
  - Otherwise firing is stalled
  - Upon completion, all output tokens are produced simultaneously
- Actor would also have to defer firing if another execution of the actor is running (auto-concurrency)

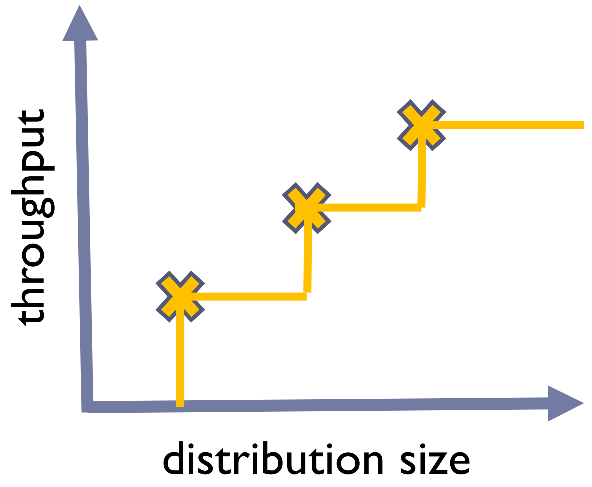
Tradeoff Analysis Based on SDF Operational Semantics



Throughput vs. Total Buffer Size (model-based)

- For a given set of buffer sizes β, throughput can be obtained by considering the firing, stall and resume conditions.
- Throughput vs. total buffer size of an SDF graph can be evaluated using Stuijk et al.'s Pareto point exploration algorithm.



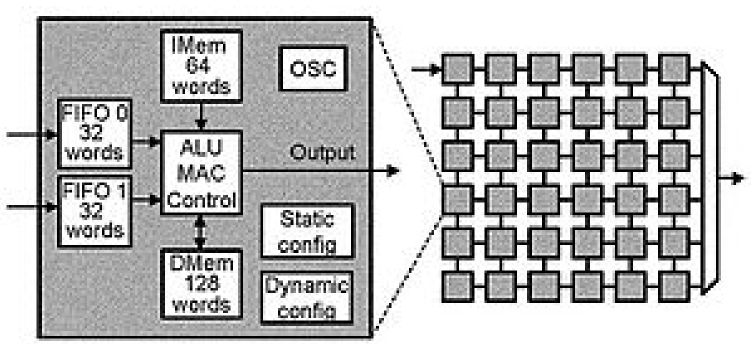


Stuijk et al. judiciously select a subset of possible $\beta(ab, bc, ac)$ values to explore

$$\beta(ab, bc, ac) = (60{,}50{,}20) \xrightarrow{\text{wrt actor 'c'}} \tau = {}^{1}/_{(1100 - 300)}$$

MPSoC Software Implementation

- Target platform: a distributed-memory message-passing system with logical direct inter-processor FIFO buffers
  - directly implemented in some platforms such as AsAP and TILE64 static network
  - logical view can also be implemented on shared memory platforms
- Tasks implemented as software modules running on parallel processors



Block diagrams of a single AsAP processor and the 6x6 AsAP 1.0 chip [Baas et al.]

Abstract View of Implementation

- Sequence of reads followed by actor's data transformation computation and finally sequence of writes to output buffers.
  - Unlike simultaneous reads (writes) assumed in the model
- Interconnect networks have limited bandwidth and in practice, each token may need to be split into $s = \left\lceil \frac{sizeof(token)}{sizeof(packet)} \right\rceil$ packets and transferred.

(A)

```
// task 'a' on P1
token ab[20];
token ac[10];

while(){
 a(ab,ac);
 write(ab,20,P2);
 write(ac,10,P3);
}
```

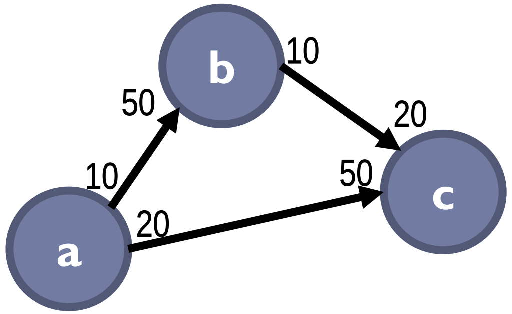
```
// task 'b' on P2
token ab[50];
token bc[10];

while(){
 read(ab,50,P1);
 b(ab,bc);
 write(bc,10,P3);
}
```

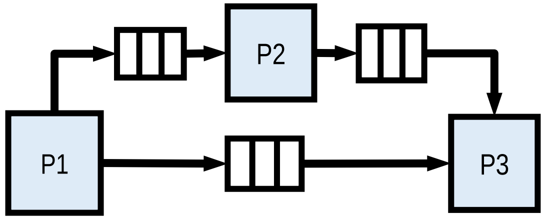
```
// task 'c' on P3
token bc[20];
token ac[50];

while(){
 read(bc,20,P2);
 read(ac,50,P1);
 c(bc,ac);
}
```

(B)

```
void write (token* x, int n, int dst){
 for i=[0,n)
  for j=[0,s)
   writePacket(x[i],j,dst);
}
```

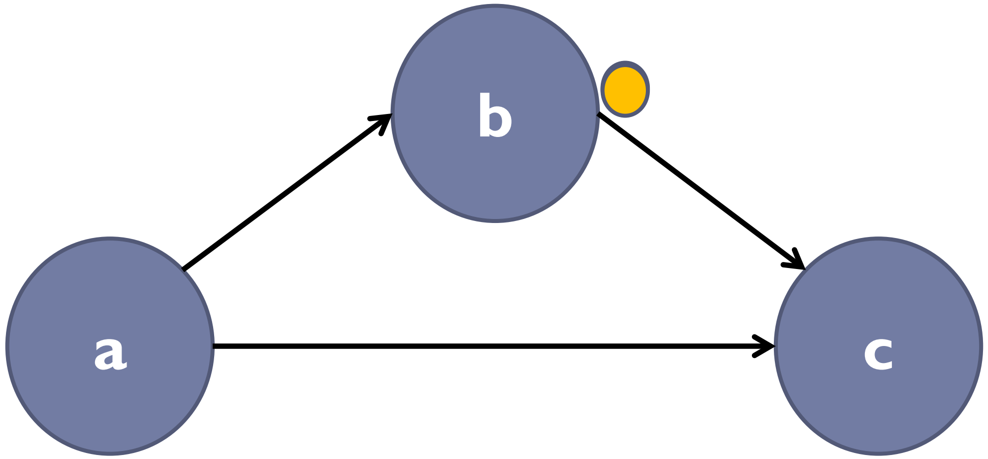
```
void read (token* x, int n, int src){
 for i=[0,n)
  for j=[0,s)
   readPacket(x[i],j,src);
}
```

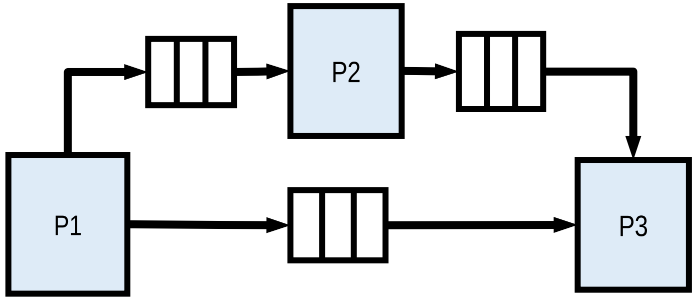




Implications of Implementation-Awareness

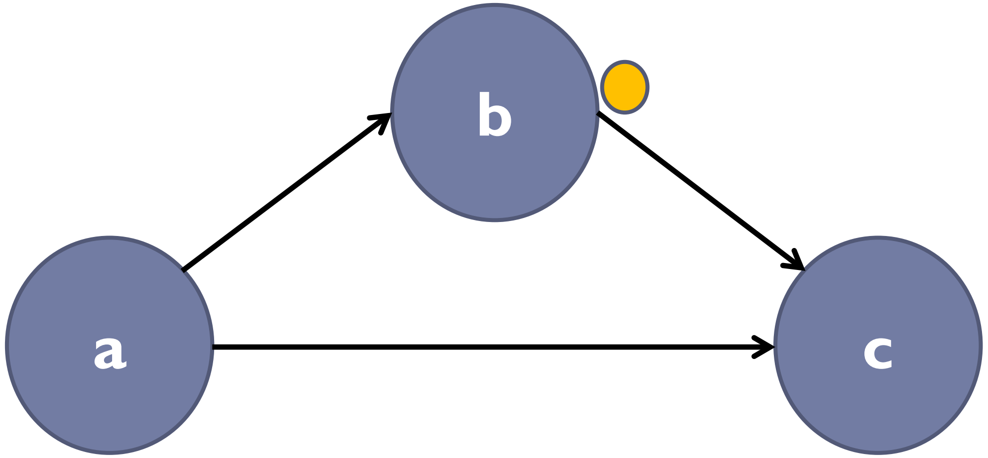
- Task can write (read) only one token to (from) only one channel at a time.
- The implementation temporal behavior diverges from the model.

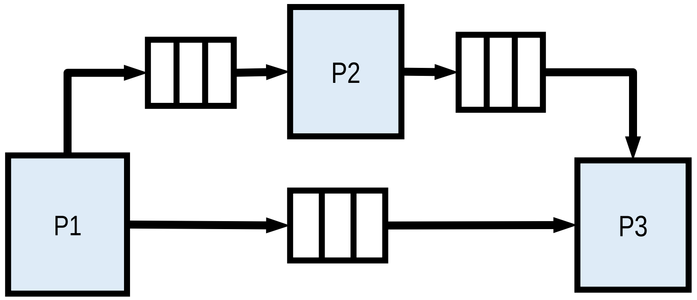




An Example Divergence in Behavior

- Task c (processor P3) stalls when it tries to read for the first time, since there is no token available on channel bc.
- Once task b (processor P2) places the first token on this channel, the stalled readPacket function in c resumes execution and reads that token.
- Observation: In this setting $\beta(bc) = 1$ would be sufficient to achieve the same throughput
  - In contrast with model-driven lower-bounds to avoid deadlock!





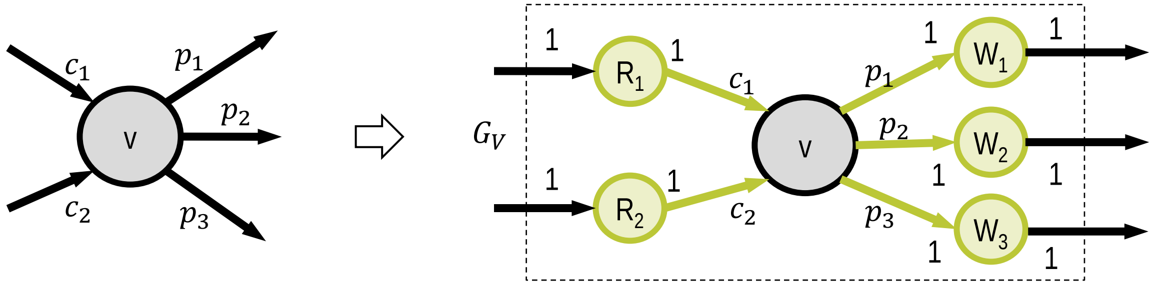
Implementation-Aware SDF Graph Transformation

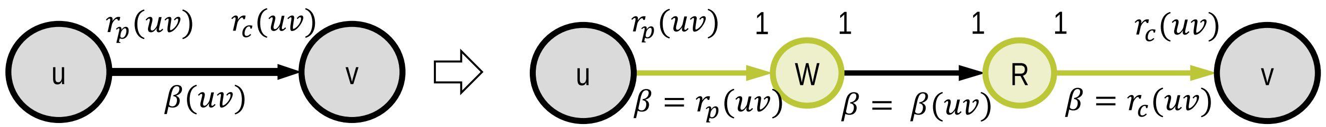
- Our proposal, a two step approach:
  - Embed limited information about target implementation into the graph
  - Analyze the transformed SDF graph G' by using the existing implementation-oblivious analysis technique (e.g., Stuijk et al. algorithm)

- Specifically, in case of the target MPSoC Implementation
  - Tasks can read (write) only one token at a time
    - Modeled by adding virtual **reader** and **writer** actors
  - Tasks can read (write) from (to) only one channel at a time
    - Modeled by adding virtual **sync** actors

Virtual Reader and Writer Actors

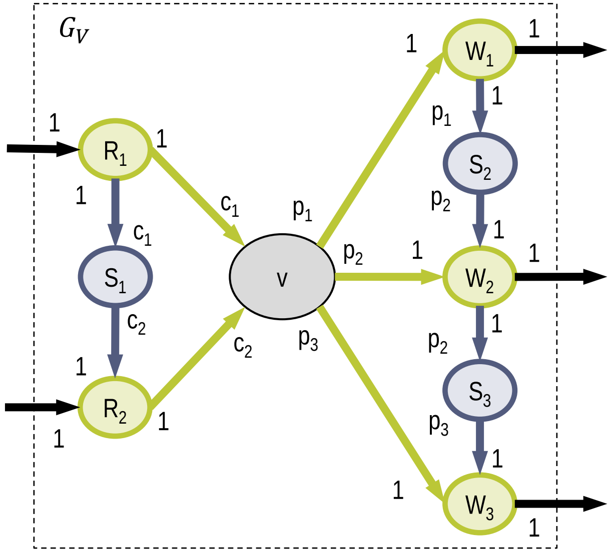
- Reader and writer actors
  - Unit data production and consumption rates
  - Identity data transformation functionality
  - For every firing of u, the writer actor fires $r_p(uv)$ times sequentially to consume the tokens produced by u.
  - For every firing of v, the reader actor fires $r_c(uv)$ times sequentially to produce the tokens needed by v.

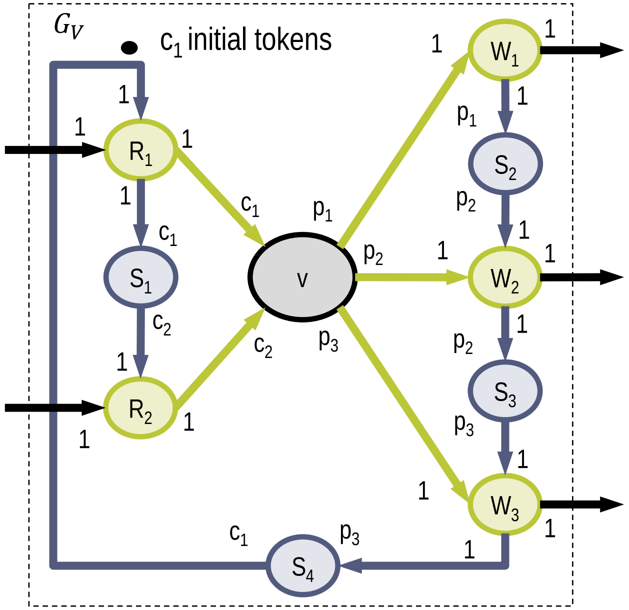




Virtual Sync Actors

- Reader and writer actors can potentially fire simultaneously.
  - Has to be eliminated to correctly model the sequential nature of task execution
- Virtual sync actors enforce the sequential order
- A sync actor between $W_{|Out(v)|}$ and R_1 to prohibit concurrent execution of reader and writer actors



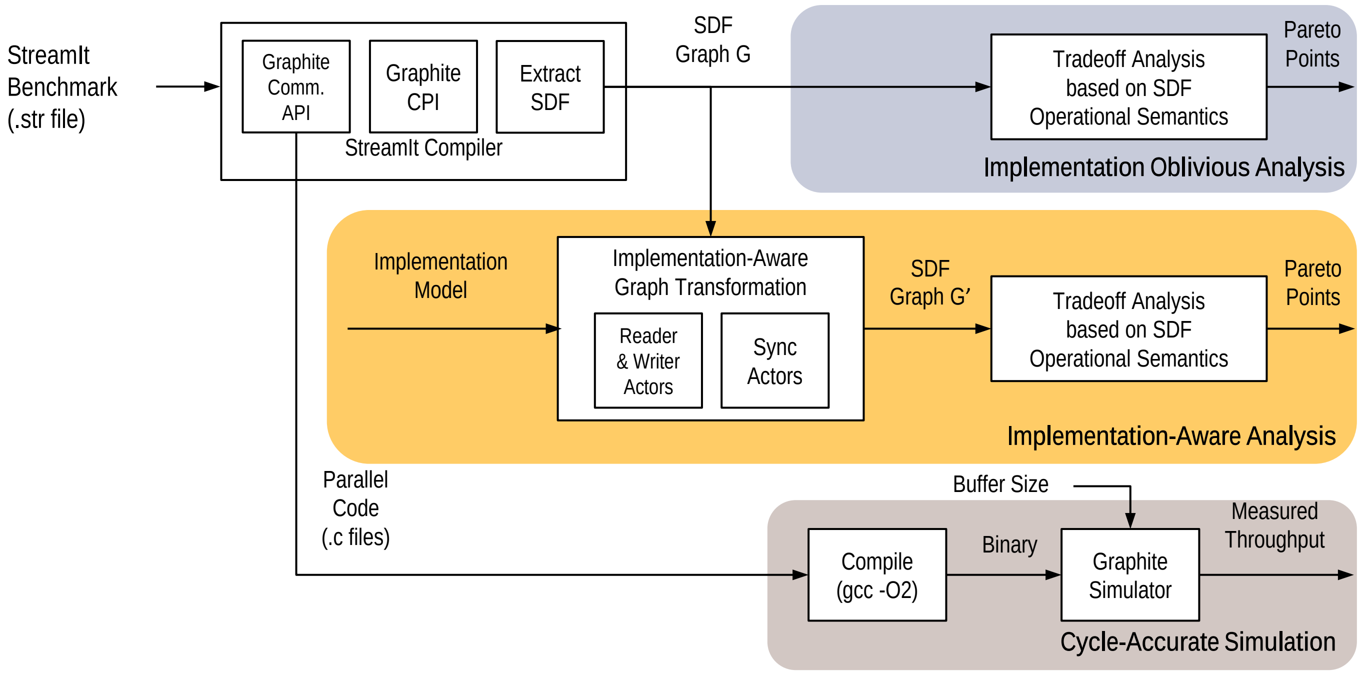


Impact on Throughput

- Read, write and sync actors are added to model the sequential order among read and write operations in the implementation
  - must not have any impact on the total execution time of the graph
- We set the execution times of reader and writer actors to zero, and assign the entire execution time of the original actor to v.
  - If specific parameters of the target architecture are known, the model fidelity could be improved by breaking down the actor latency between read/write and data transformation operations.
- A number of properties are proved about the proposed transformation
  - Examples: lower bounding memory requirement and asymptotic throughput
  - Please refer to the paper

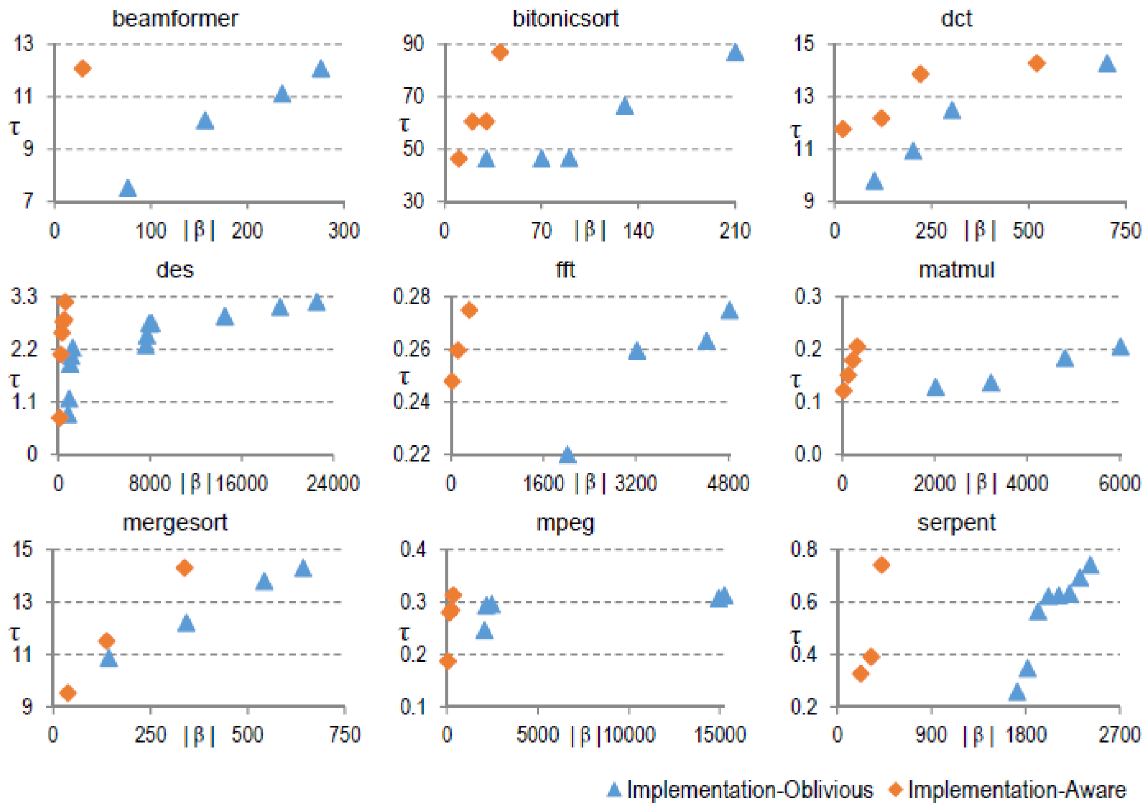
Experiments

- Evaluated using StreamIt benchmarks.
- SDF graph, data rates $(r_p \; and \; r_c)$ and estimates of actor execution time (ε) are extracted using StreamIt compiler.



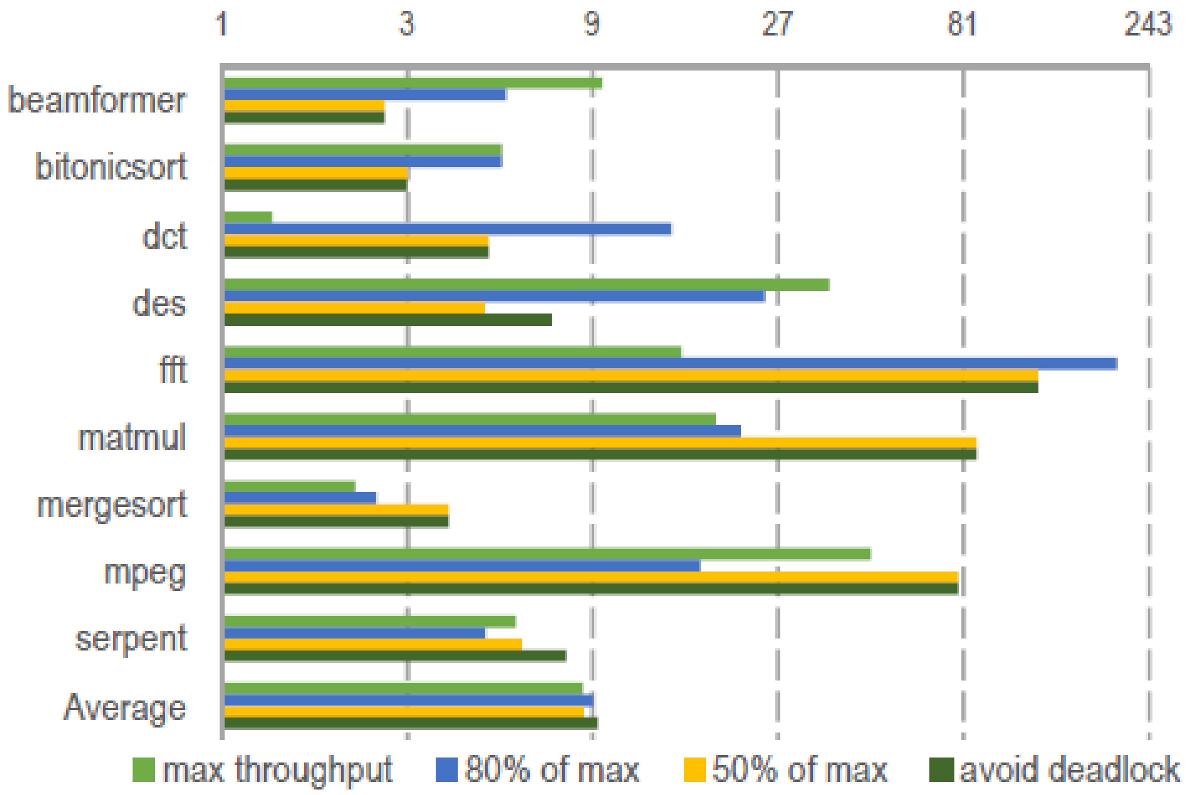
Throughput-Buffer Size Tradeoff

- The analysis yields a set of pareto optimal points between the total buffer size, $|\beta|$, and the corresponding overall throughput, τ.
- The implementation-aware tradeoff analysis yields substantially smaller buffer estimates compared to the implementation-oblivious analysis for the same level of throughput.



Total Buffer Size Reduction

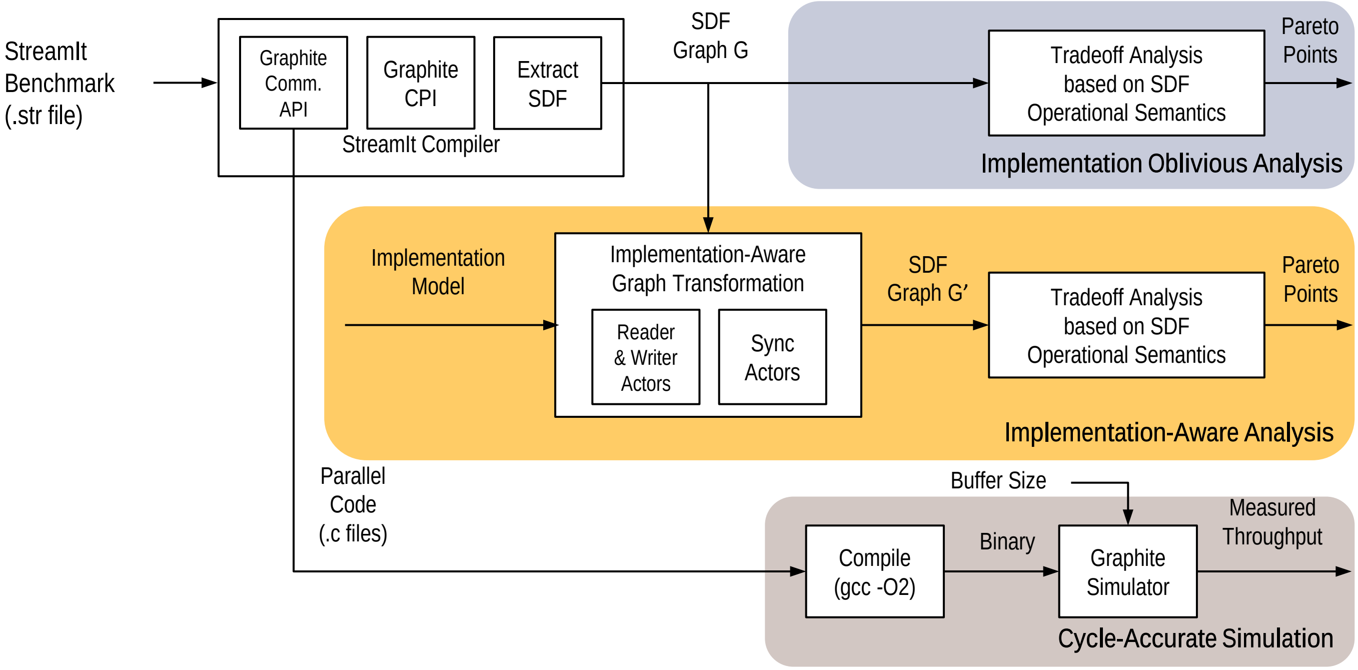
- Implementation-aware analysis yields a substantial reduction in total buffer size requirement, under throughput constraints.



Reduction in total buffer size estimates using implementation aware analysis.

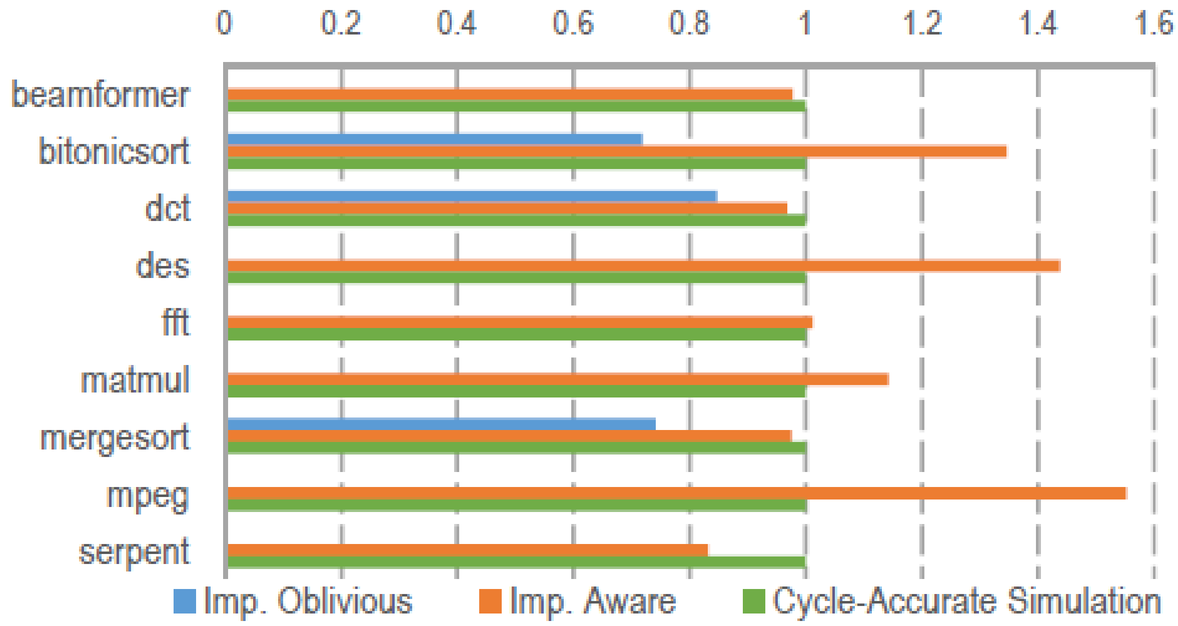
Accuracy of Model-Based Analysis

- Simulated executable binaries under different buffer sizes using Graphite Multicore simulator.
- Buffer size distribution ($\beta(uv)$ for all channels uv) adjusted to match estimates that result in the maximum throughput according to implementation-aware model analysis.



Accuracy Comparison of Throughput Estimates

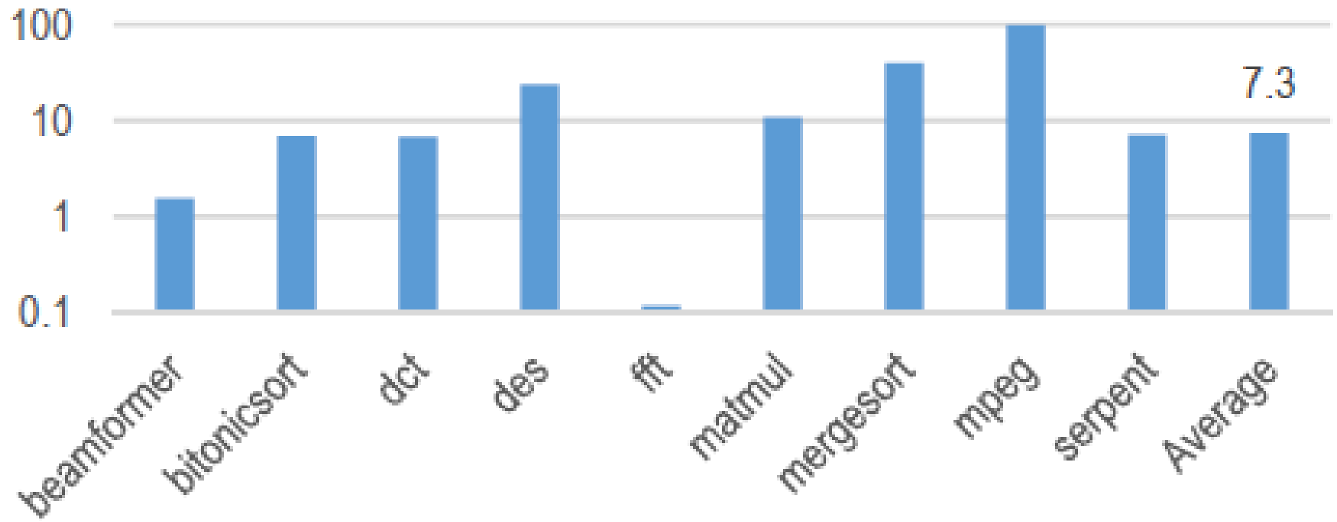
- Implementation-oblivious analysis vs. cycle-accurate simulation
  - The implementation oblivious analysis falsely reports deadlock in six out of nine benchmarks. Average error: 74%.
- Implementation-aware analysis vs. cycle-accurate simulation
  - Throughput estimation error is less than 5% in beamformer, dct, fft and mergesort. Average error: 19%



Throughput estimates normalized relative to cycle-accurate simulation results

Impact on Analysis Runtime

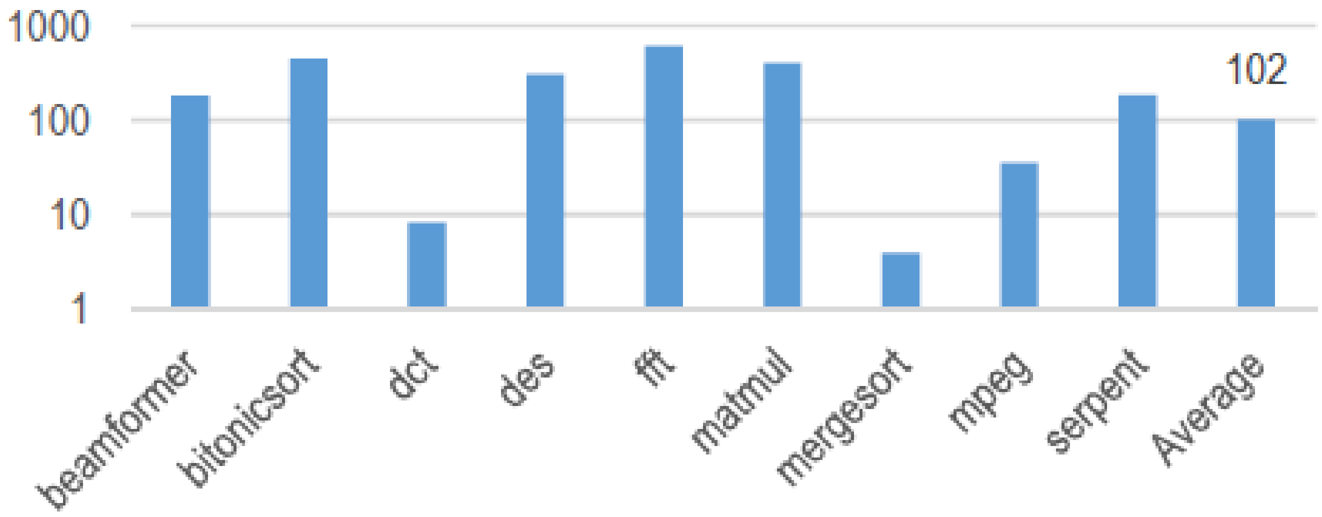
- While heavily application-dependent, the execution time of implementation-aware analysis is on average about 7.3 times of baseline analysis.
- Execution time overhead is mostly due to the larger graph complexity (reader, writer and sync).



Runtime of implementation-aware relative to implementation-oblivious model analysis.

Impact on Analysis Runtime

- For 6 out 9 benchmarks, implementation-aware analysis runs more than 2 orders of magnitude faster than simulation.
- On average, it takes about 102X longer to run cycle-accurate simulations than to run the proposed implementation aware analysis.



Runtime of cycle-accurate simulation relative to the proposed technique.

Thank you

Questions