Static Translation of Stream Program to a Parallel System

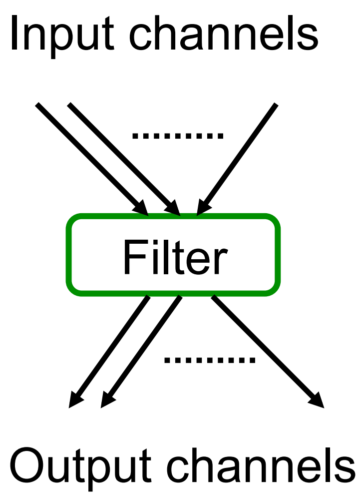
S. M. Farhad
The University of Sydney

The Free Lunch Is Over

- Uniprocessor hits the physical limit
  - Multicore, many core systems
- Programming challenges now
  - Multiple flows of control and memories
  - Finding algorithm that can run in parallel
  - Correctness of the program
  - Synchronization Issues
  - Debugging the program

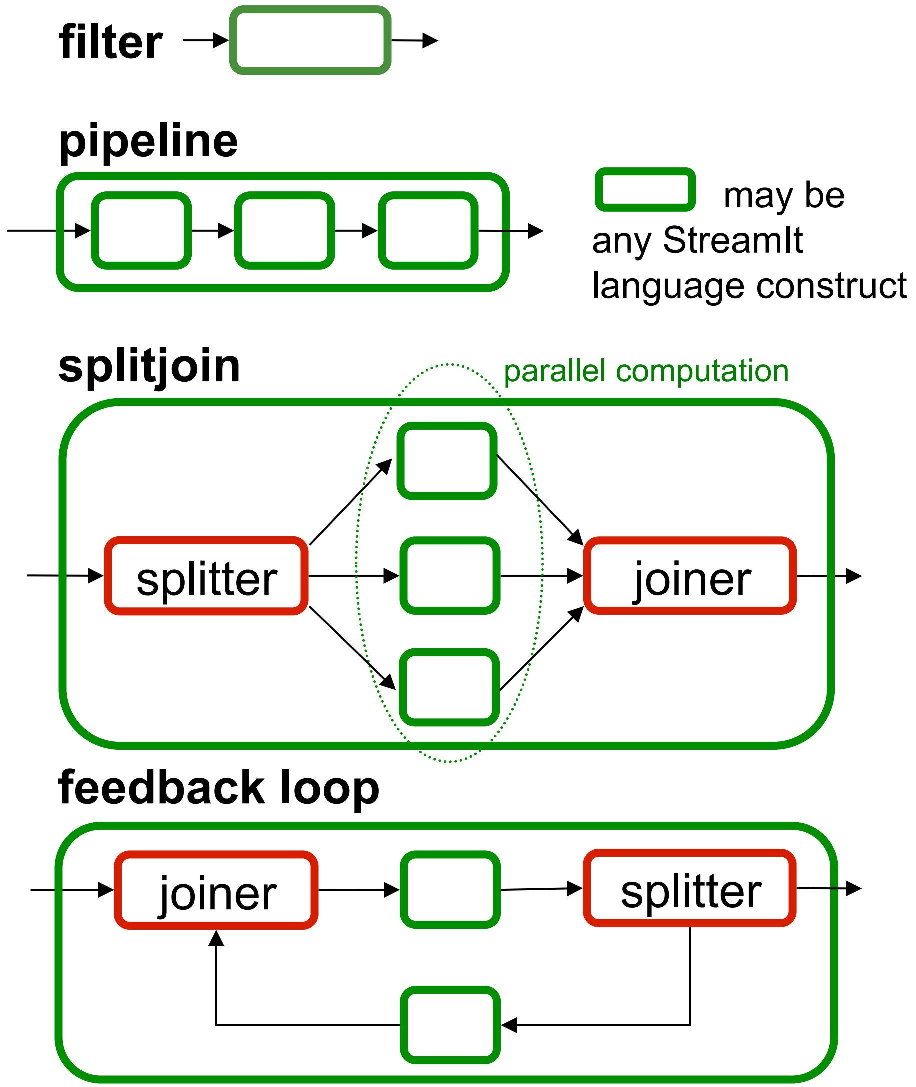
Stream Programming Paradigm

- It expresses parallelism inherently
- It leverages program structure to discover parallelism and delivers high performance
- It is structured around notion of a "stream"
- A streaming computation represents
  - A sequence of transformations on the data streams
  - A stream program is the composition of filters into a stream graph



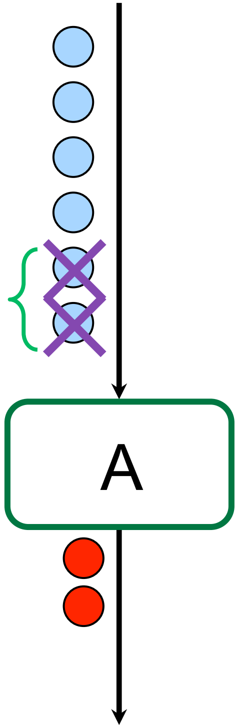
StreamIt Language

- An implementation of stream programming
- It exposes parallelism
- It is architecture independent
- Modular



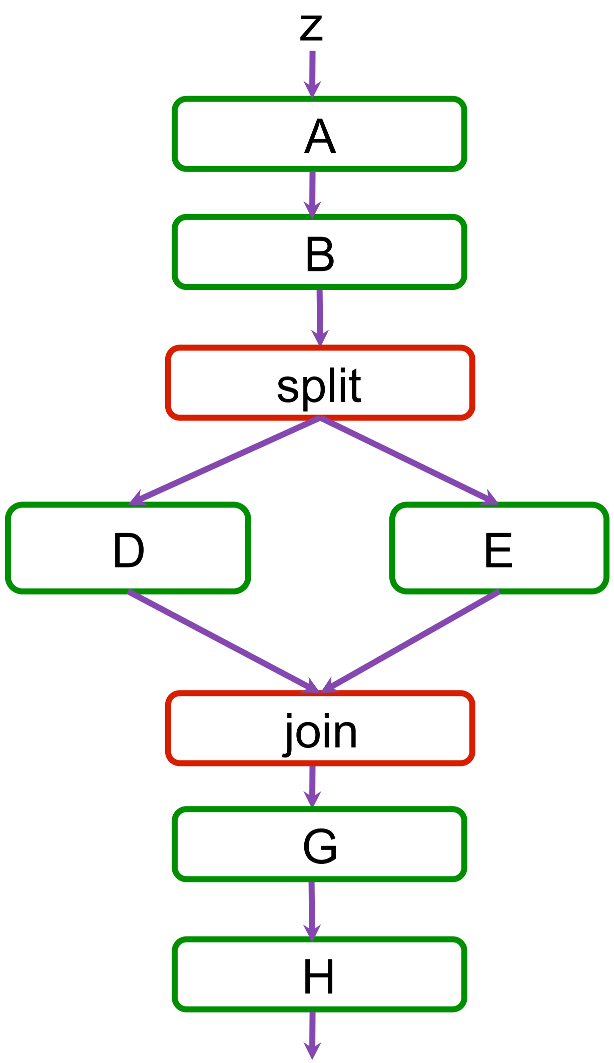
A Simple Filter in StreamIt

```
int->int filter A {
    init {
        // empty
    }
    work pop 1 peek 2 push 1 {
        push(peek(0) + peek(1));
        pop();
    }
}
```



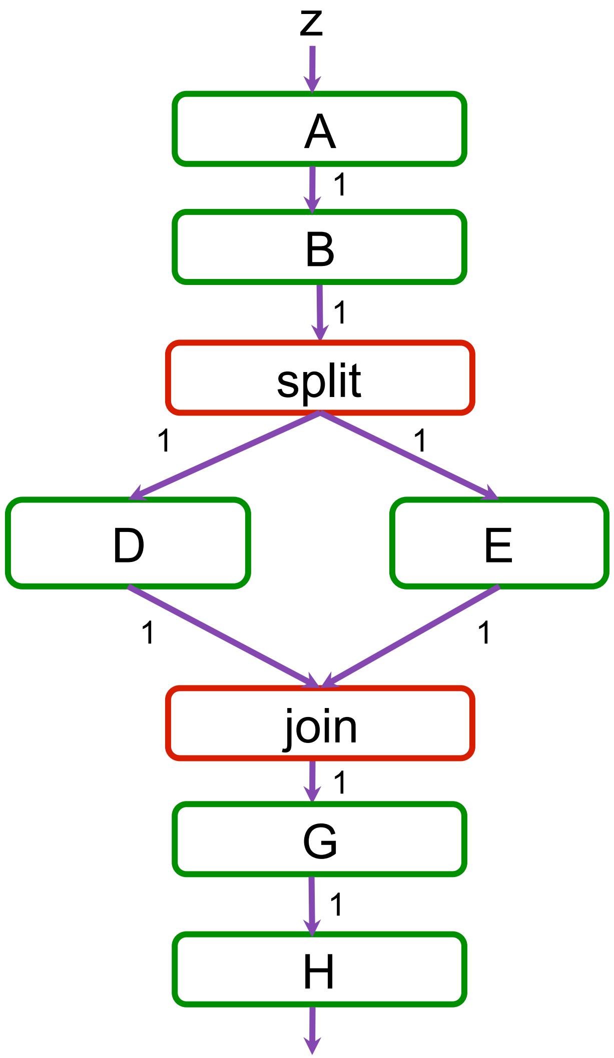
StreamIt Example

```
float->float pipeline Example() {
    add A();
    add B();
    add splitjoin {
        split duplicate;
        add D();
        add E();
        join roundrobin;
    }
    add G();
    add H();
}
```



Research Question?

- How to parallelize the stream program
  - Optimize throughput
- Synchronous model
  - Describes bandwidths of streams in steady state
- Bottleneck actor
  - An actor constrains “z”

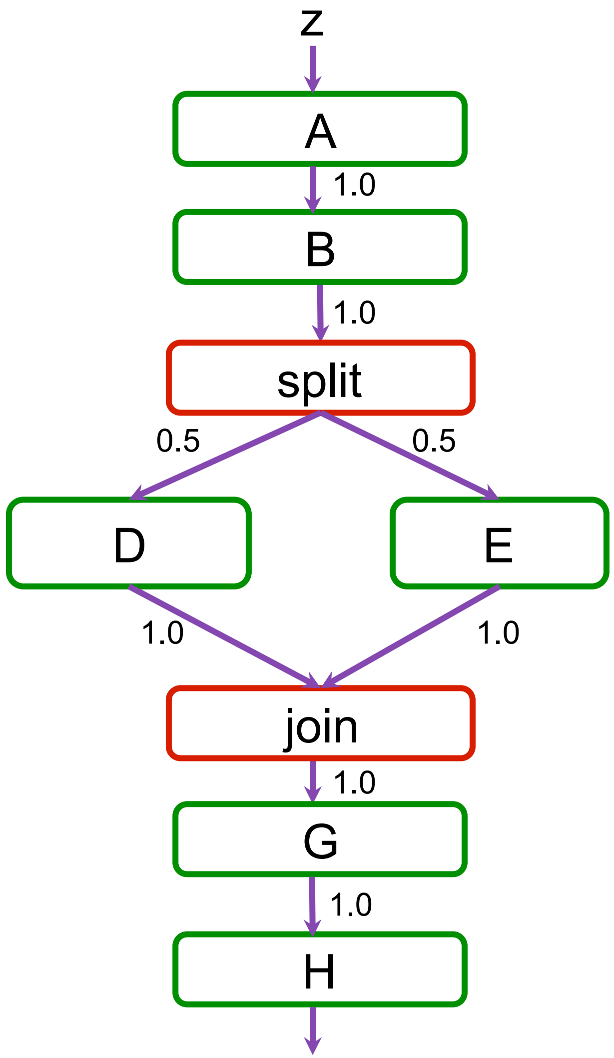


Static Translation of Stream Programs

- We propose
  - A simple quantitative analysis to resolve bottlenecks in stream programs
  - Actors mapping technique to a parallel system
  - Scheduling of mapped actors in each processor
- Our goal
  - To statically optimize the throughput of a stream program

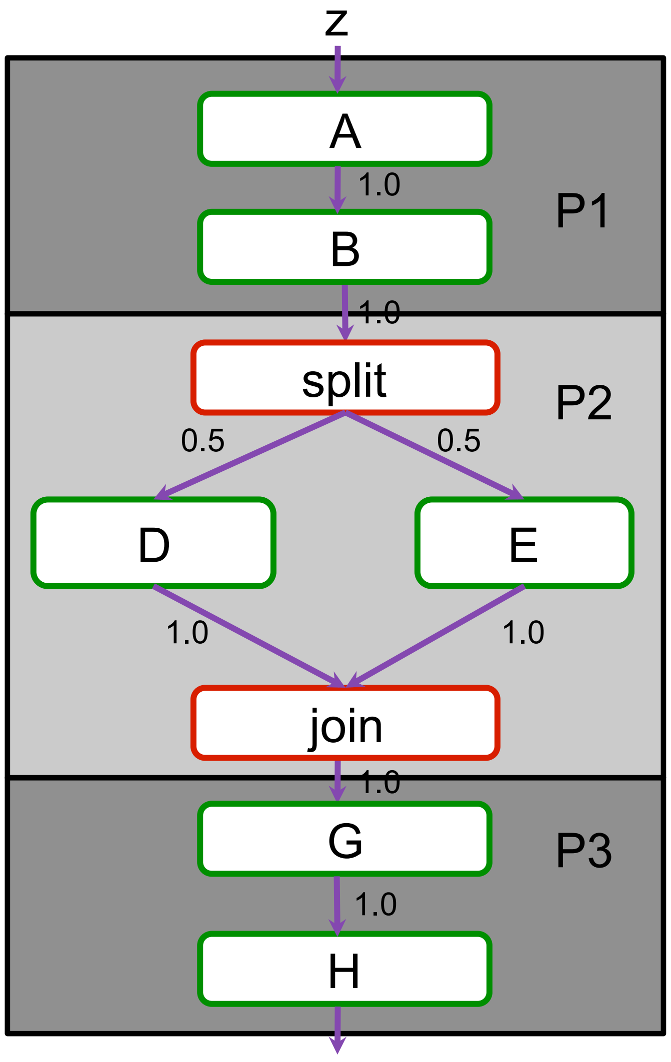
Finding Closed Form

- We assume bandwidth functions are "linear functions"
- We have a system of simultaneous linear equations
- The output are shown as weights
- We express output of each actor as a function of "z"



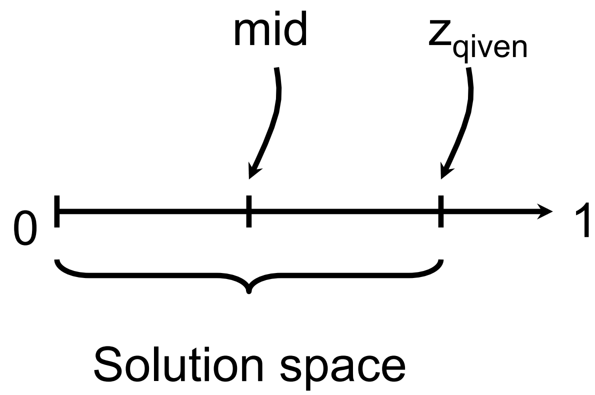
Mapping Actors to Processors

- An NP-hard problem
  - Takes too long time
- Approximation algorithm
  - Much faster *O(n log n)*
  - Non-optimal solution
- We formulate Integer Linear Programming problem considering
  - Input, output and processing bandwidths of each actor
  - Processing capacity each processor
  - Inter processor communication bandwidths



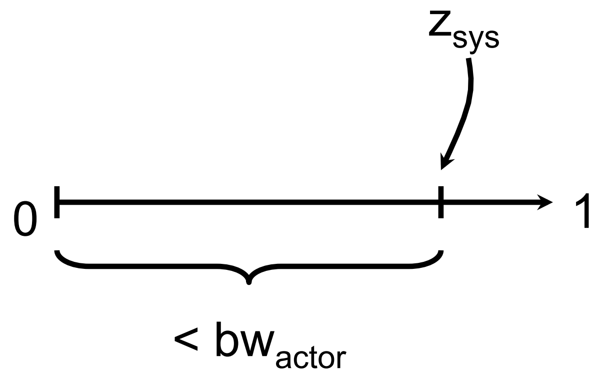
Mapping Actors to Processors Condt.

- We develop a test for a given "z"
- We find a mapping using "Binary Search"

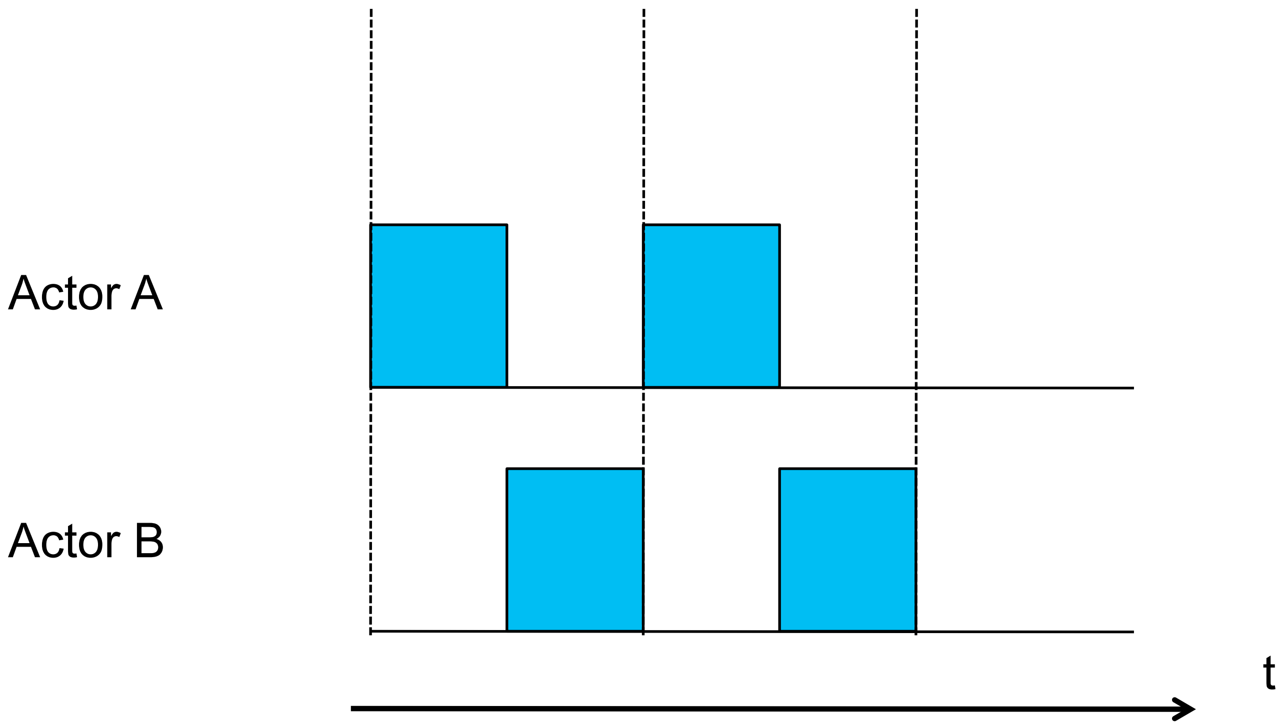


Resolve Bottleneck

- We find bottleneck by quantitative analysis
  - Actors constraining the system bandwidth
- Duplicate hot actors
- Then we run the whole process again
- Reason of considering bottleneck after mapping

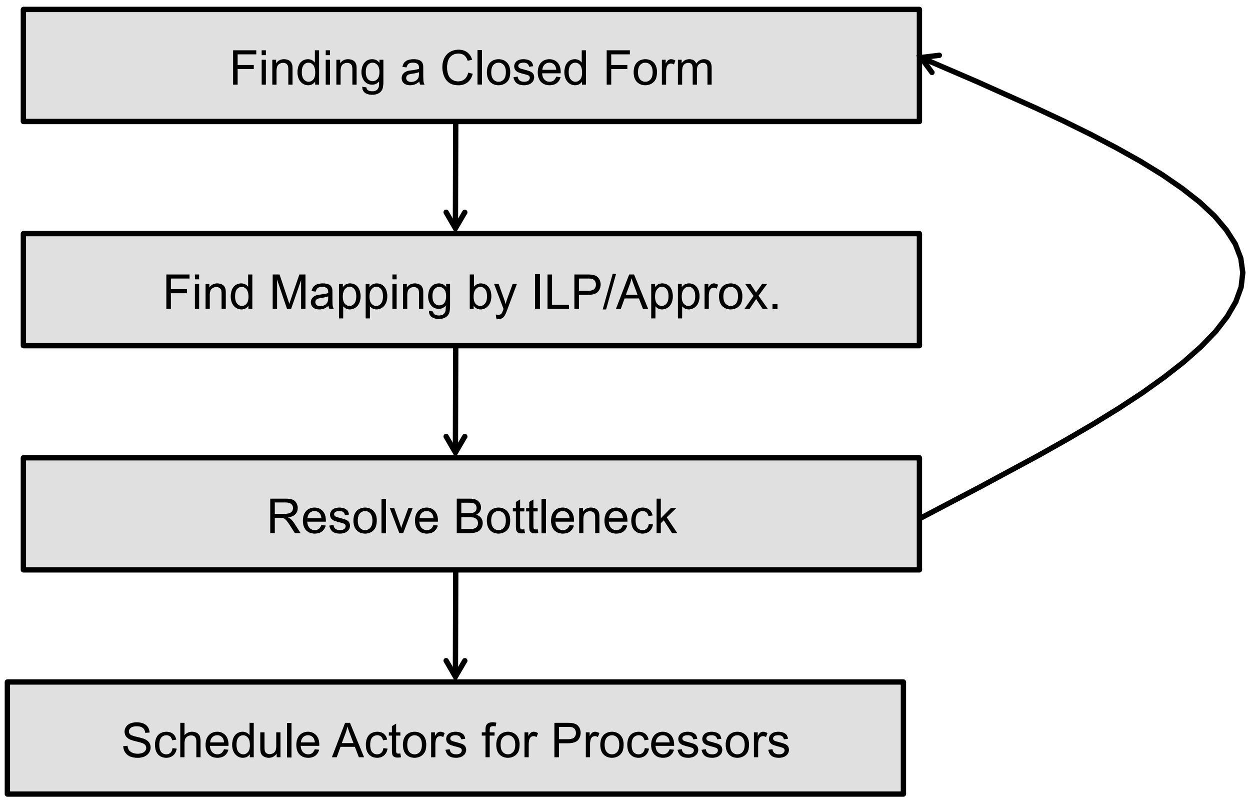


Actor Scheduling for Processors

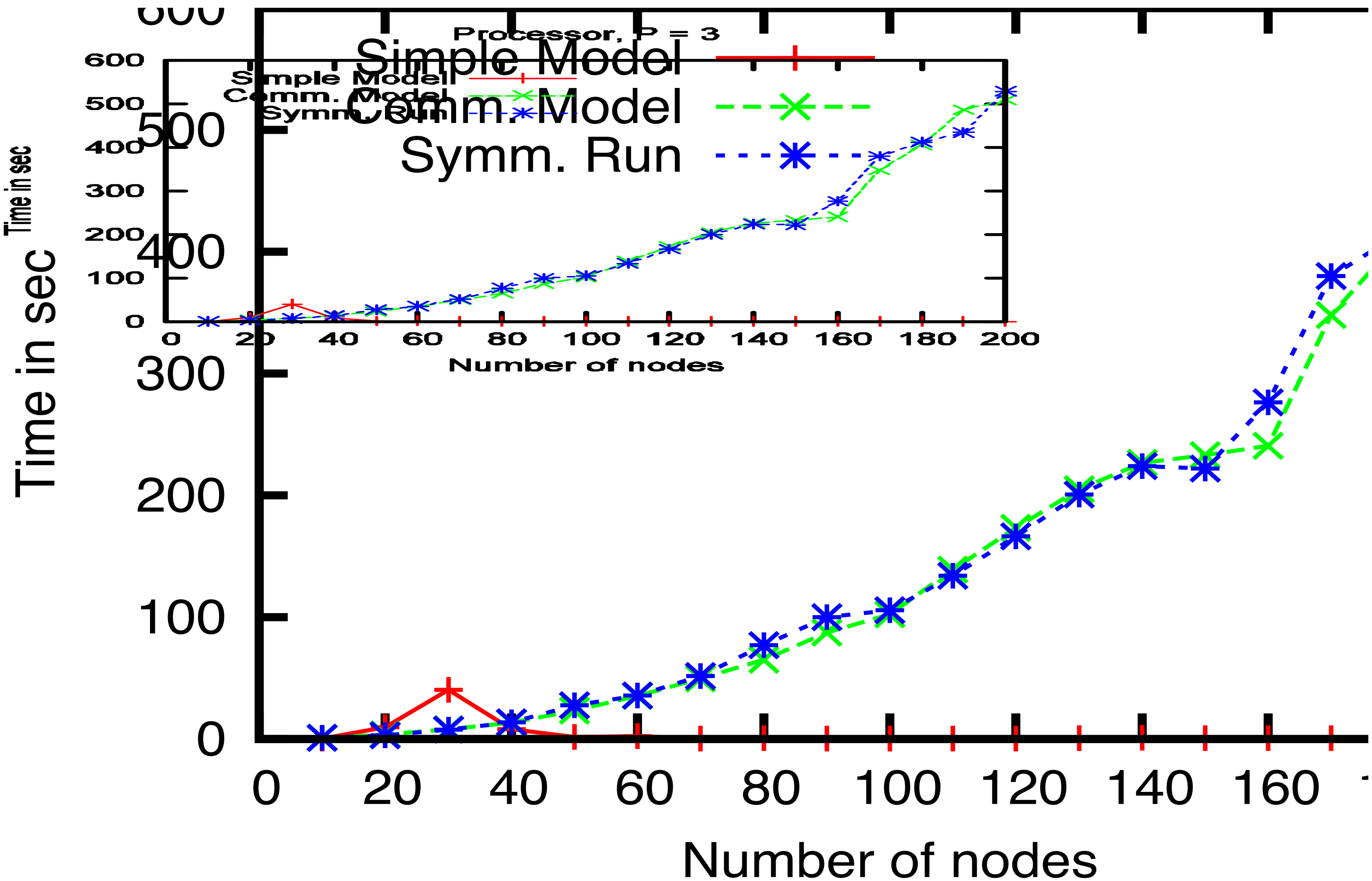


Schedule of processor P_1

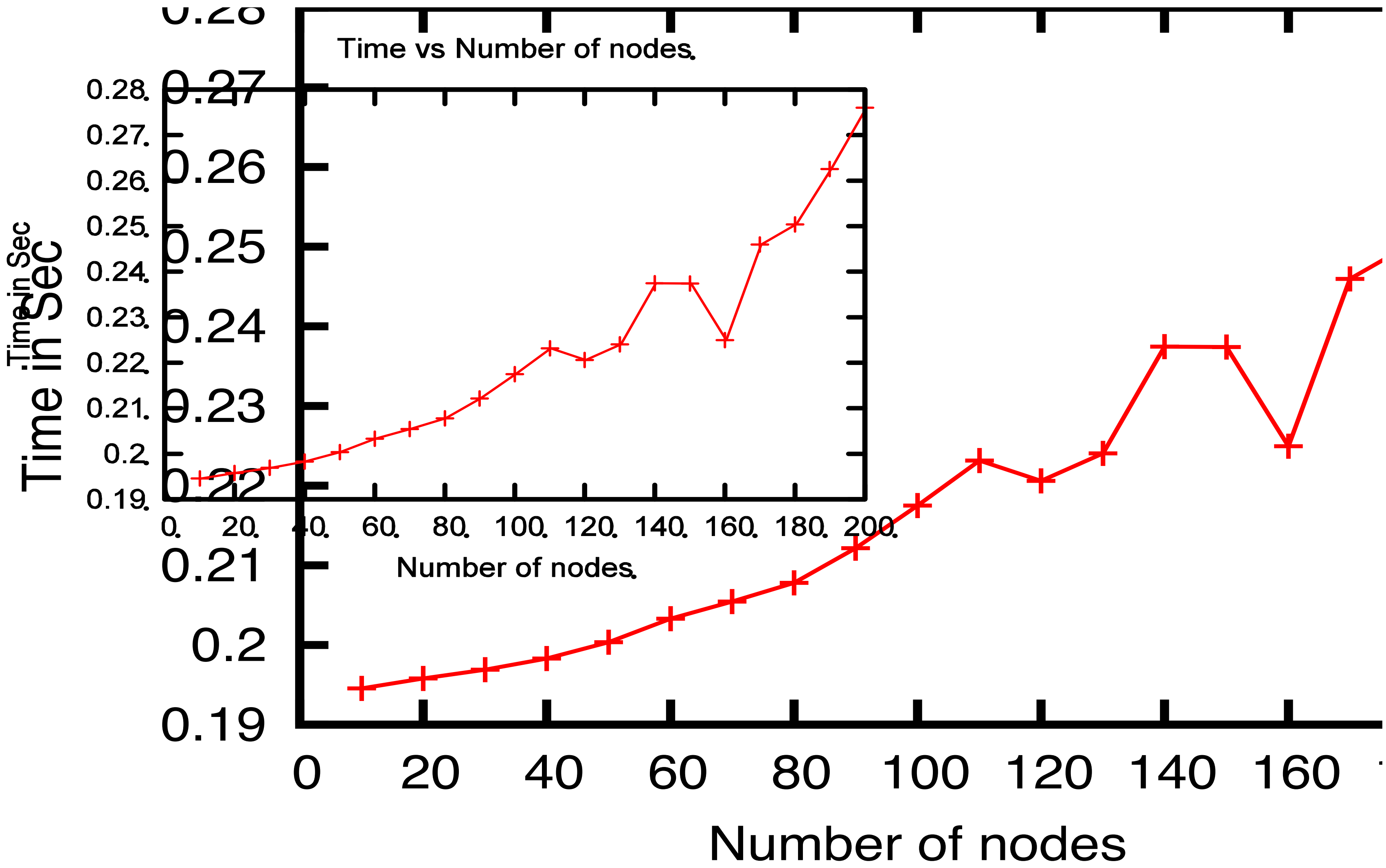
Entire Framework



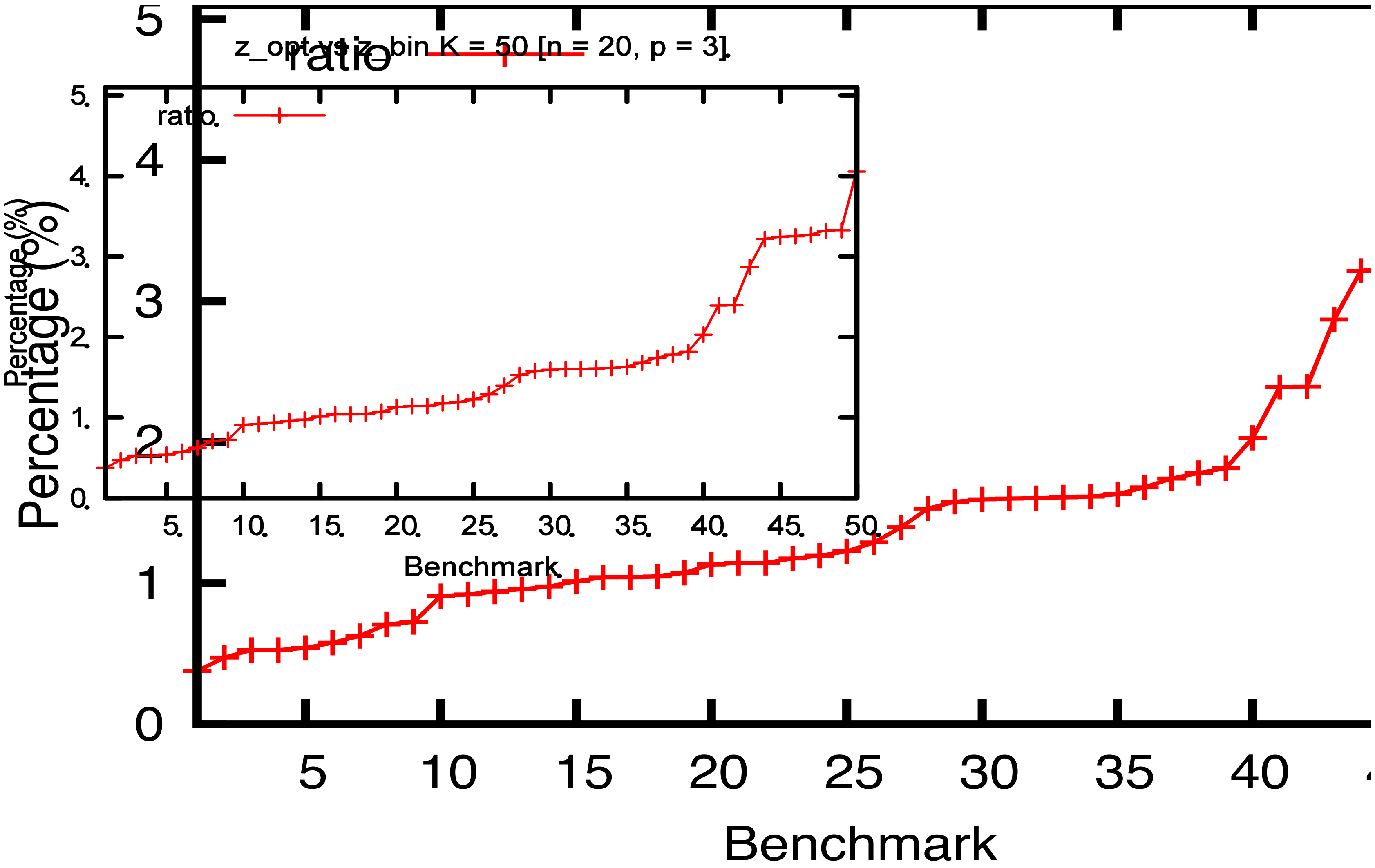
Execution Time of ILP Solver



Execution Time of a Variation of Bin-packing Algorithm



Optimal vs. Approximation



Summary

- We develop a synchronous model for stream programs
- Statically optimize the throughput of stream programs
- Resolving bottleneck by simple quantitative analysis
- Finding an approximation for the mapping problem

Related Works

[1] Static Scheduling of SDF Programs for DSP [Lee ‘87]

[2] StreamIt: A language for streaming applications [Thies ‘02]

[3] Phased Scheduling of Stream Programs [Thies ’03]

[4] Exploiting Coarse Grained Task, Data, and Pipeline Parallelism in Stream Programs [Thies ‘06]

[5] Orchestrating the Execution of Stream Programs on Cell [Scott ’08]

[6] Software Pipelined Execution of Stream Programs on GPUs [Udupa‘09]

[7] Synergistic Execution of Stream Programs on Multicores with Accelerators [Udupa ‘09]