# Bottleneck Elimination from Stream Graphs

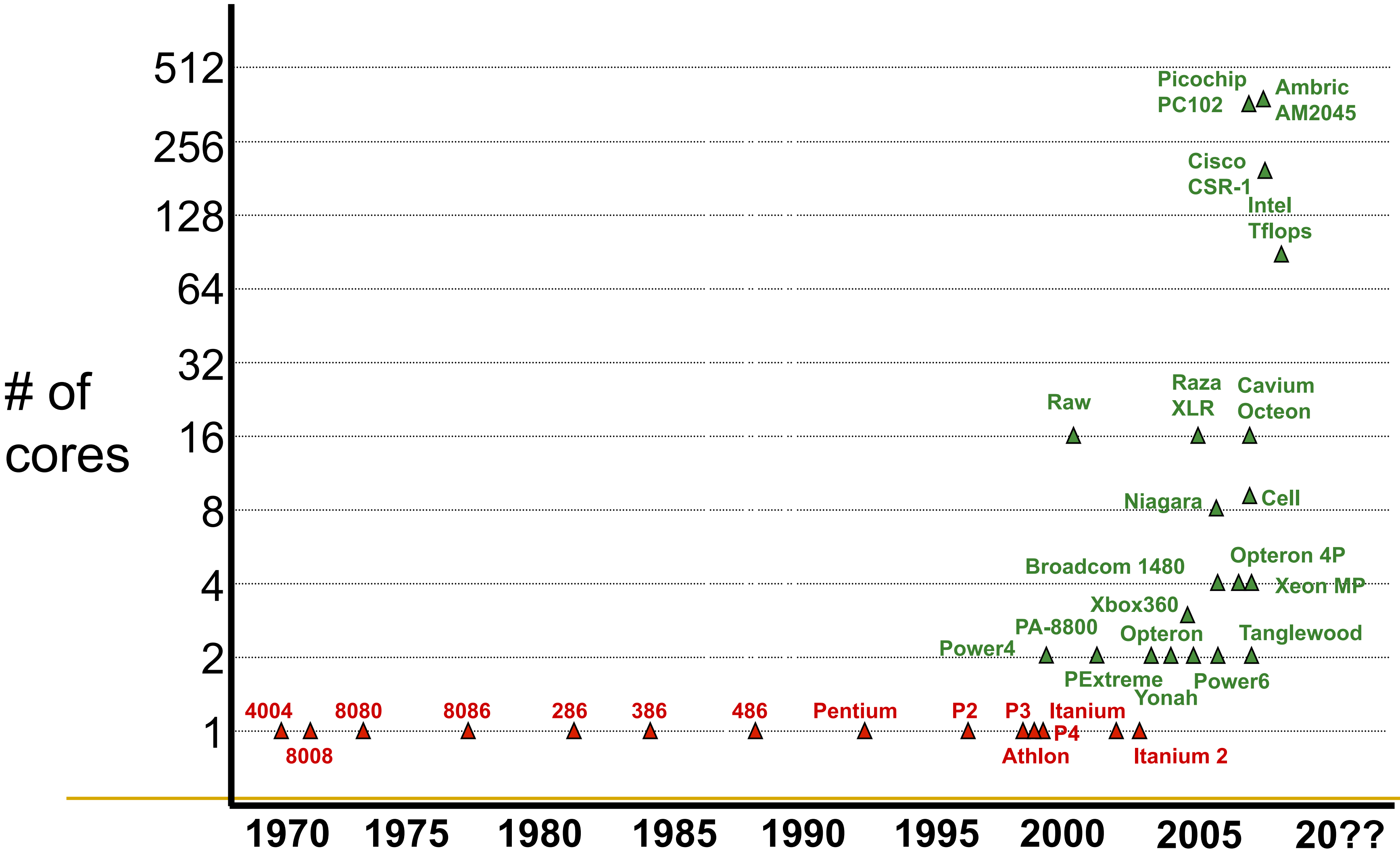
S. M. Farhad
The University of Sydney

Joint work with
Yousun Ko
Bernd Burgstaller
Bernhard Scholz

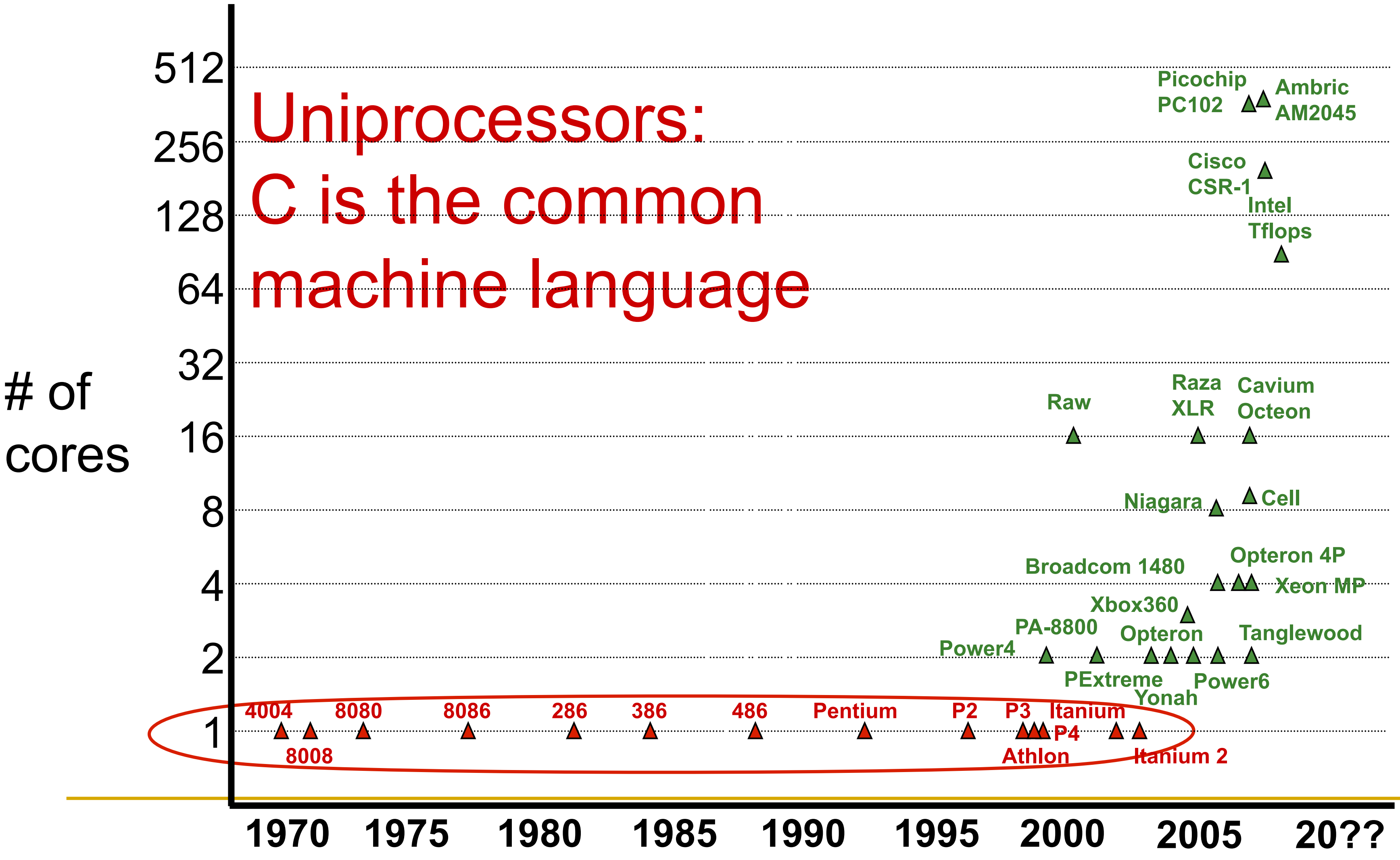
# Outline

- Motivation
  - Multicore
  - Stream programming
- Research question
- Our work
- Summary

# Multicores Are Here!

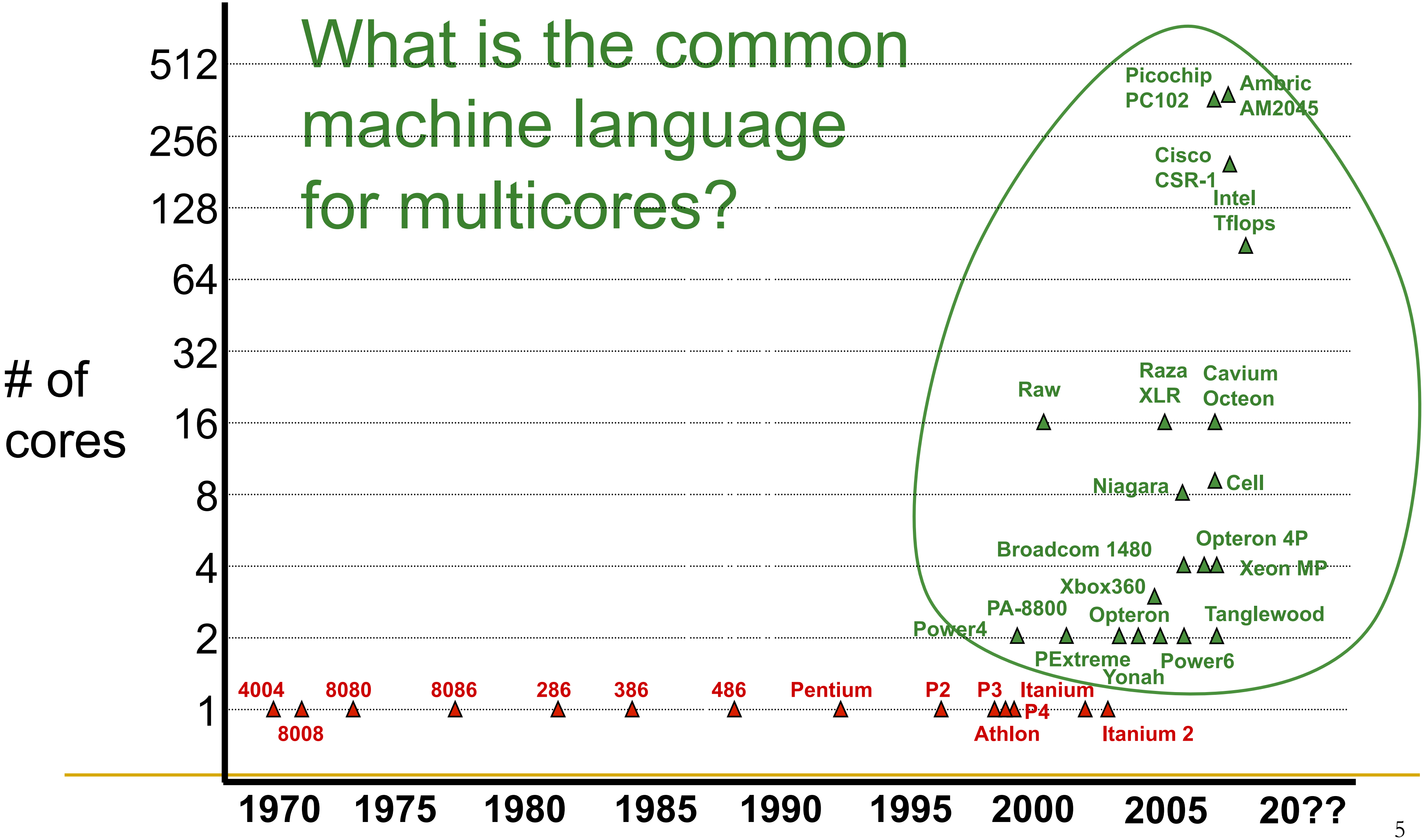


# Multicores Are Here!



Uniprocessors: C is the common machine language

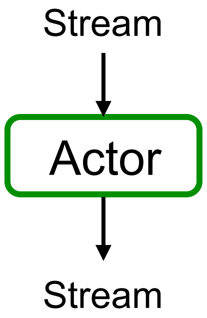
# Multicores Are Here!



What is the common machine language for multicores?

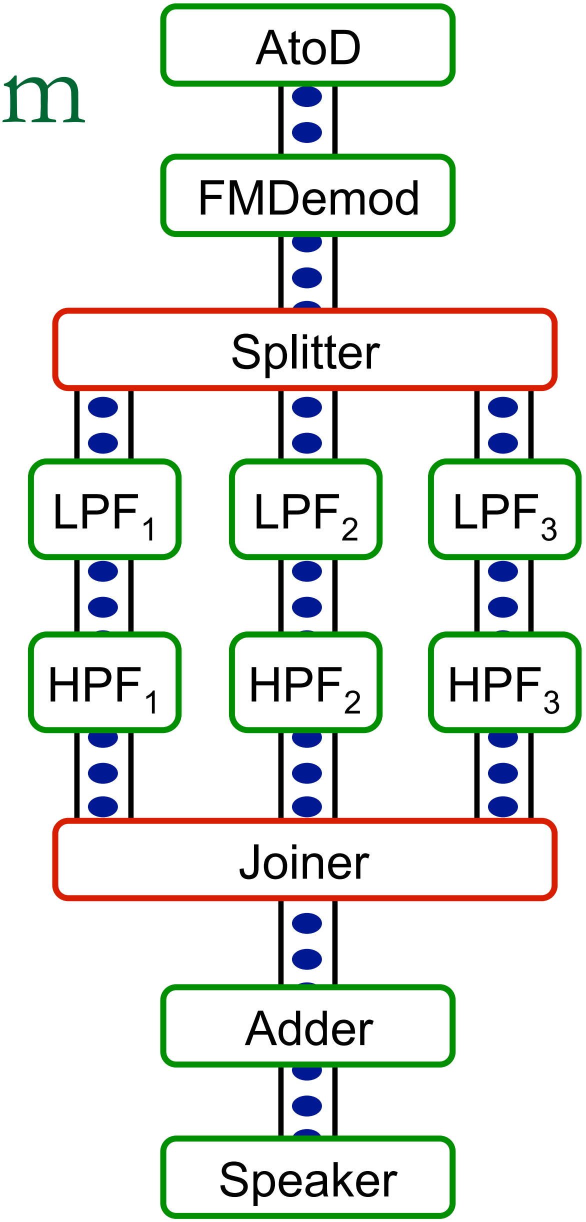
# Stream Programming Paradigm

- Research topic in parallel programming
- Various forms of parallelism
  - Pipeline, task, and data
- Applications
  - Signal Processing
  - Multi-media
  - High-Performance Computing
- Programs expressed as stream graphs
  - Streams
    - Infinite sequence of data elements (aka. Tokens)
  - Actor
    - Functions applied to streams



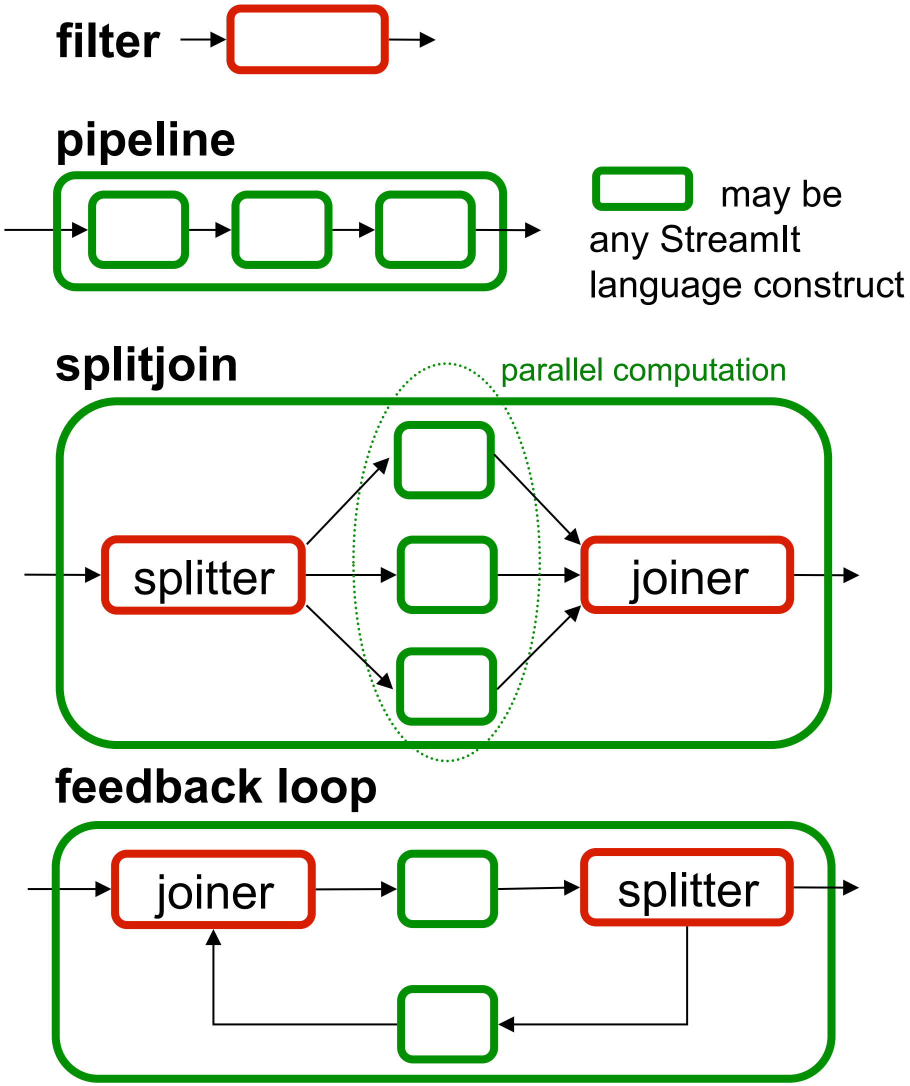
# Properties of Stream Program

- Regular and repeating computation
- Independent actors with explicit communication
  - Producer / Consumer dependencies

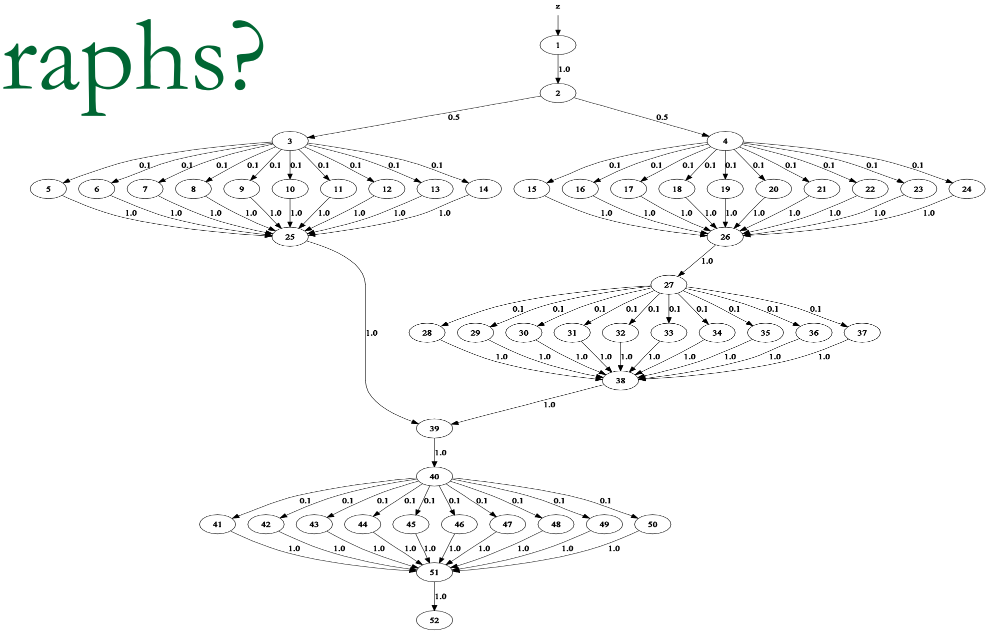


# StreamIt Language [ASPLOS'2&6, PLDI'3]

- An implementation of stream prog.
- Each construct has single input/output stream
- Hierarchical structure
- Filters can be stateful/stateless

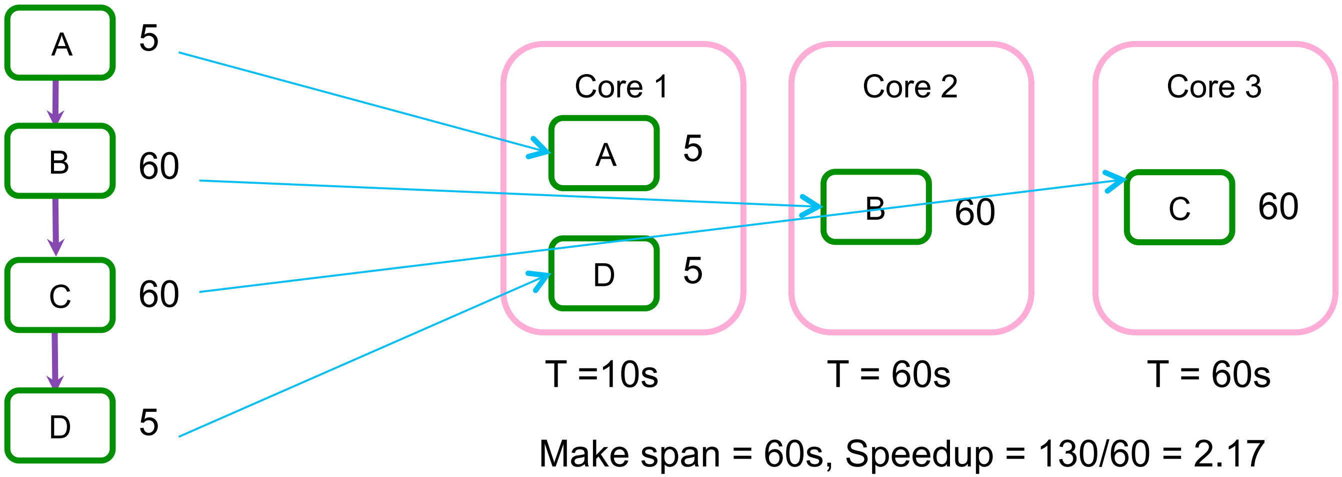


# Research Question: How to Eliminate Bottlenecks (Hot Actors) from Stream Graphs?



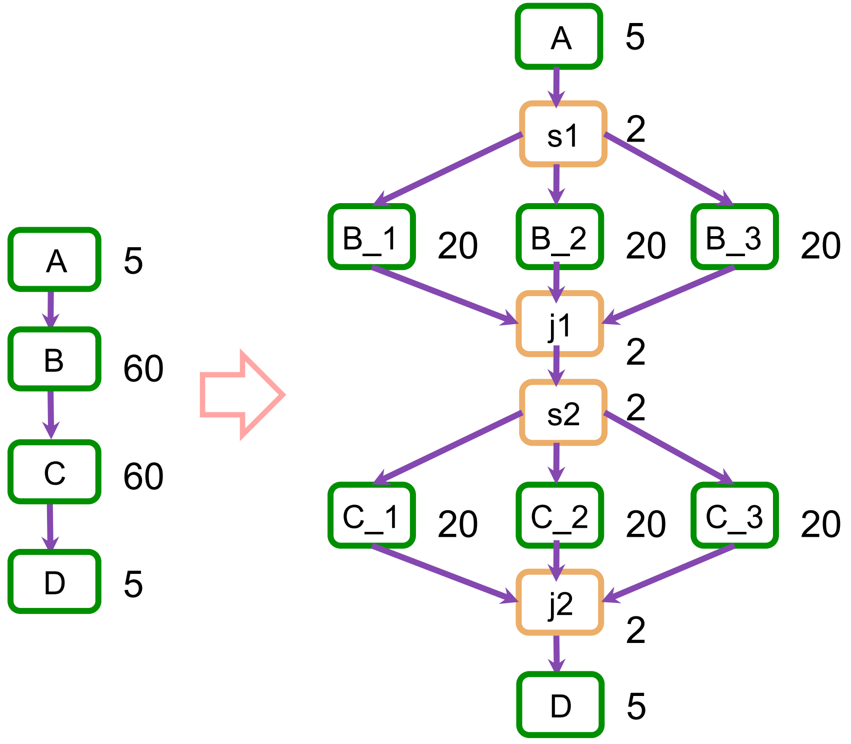
Input Graph

# Mapping Actors



Make span = 60s, Speedup = 130/60 = 2.17

# Bottleneck Actors Limit the Performance



Hot actor duplication

| Core 1 | | Core 2 | | Core 3 | |
|---|---|---|---|---|---|
| A | 5 | B_2 | 20 | B_3 | 20 |
| s1 | 2 | j1 | 2 | C_3 | 20 |
| B_1 | 20 | s2 | 2 | D | 5 |
| C_1 | 20 | C_2 | 20 | | |
| | | j2 | 2 | | |
| T = 47s | | T = 46s | | T = 45s | |

Make span = 47s, Speedup = 130/47 = 2.77

# Bottleneck Resolving of Stream Program Contd.

- Current state of the art
  - Integer Linear Programming
    - Intractable

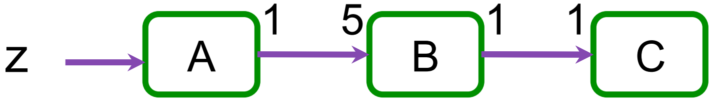
- How to find a fast and good solution?
  - Heuristics
  - Optimal

# Our Work

- A data rate transfer model to detect and eliminate bottlenecks
- We separate the bottleneck elimination from the actor allocation
- Heuristics to solve bottleneck problem efficiently

# Our Data Transfer Model

- Throughput depends on the data rate of the actors (maximize)
- Data transfer model forms a system of sim. functional linear equation
- Compute a closed form of the output data rate
- We also consider a processor utilization function for each actor



$x_A = z \qquad x_B = 0.2x_A \qquad x_C = x_B$

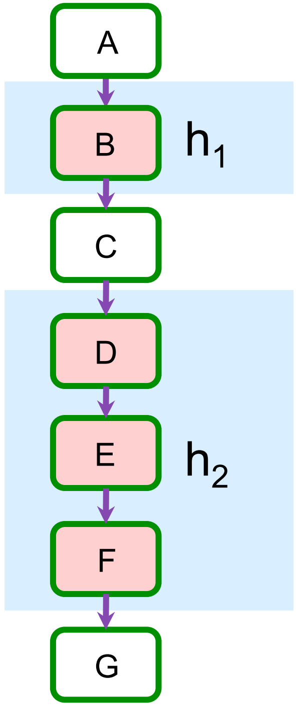
$x'_A = z \qquad x'_B = 0.2z \qquad x'_C = 0.2z$

# Bottleneck Analysis

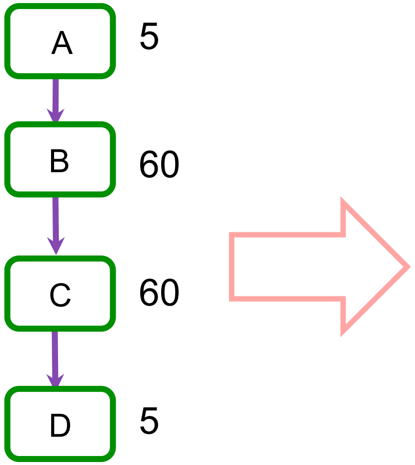
- The throughput is limited by
  - Processor capacity of the cores
  - Memory bandwidth
- A quantitative analysis determines
  - An upper bound of the throughput imposed by an actor
  - An upper bound of the throughput imposed by the parallel system
- Hot actor
  - Upper bound (actor) < upper bound (system)

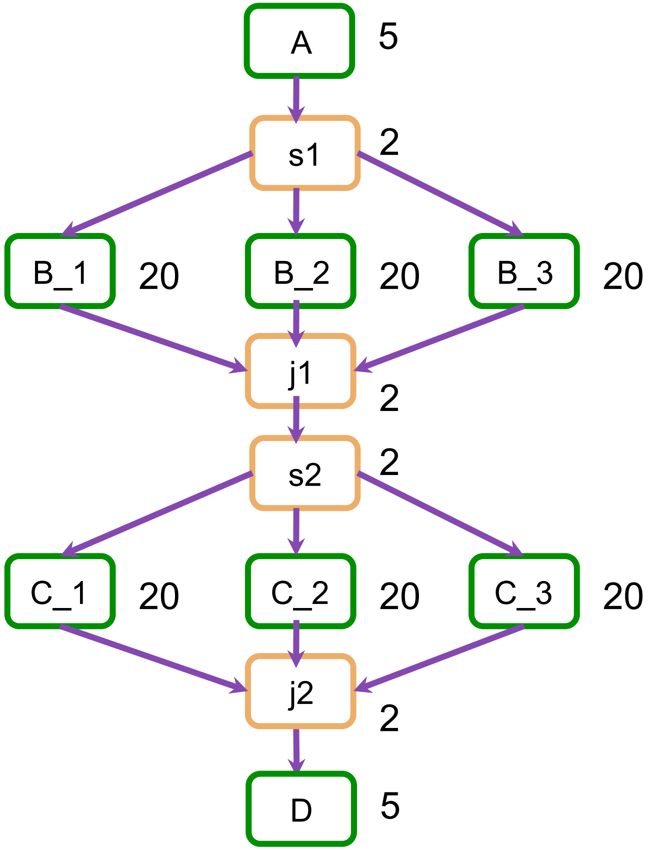
# Hot Region

- Maximal connected subgraph $h = (V', E')$ where $V' \subseteq V, E' \subseteq E$ and each $i \in V'$ is hot and stateless

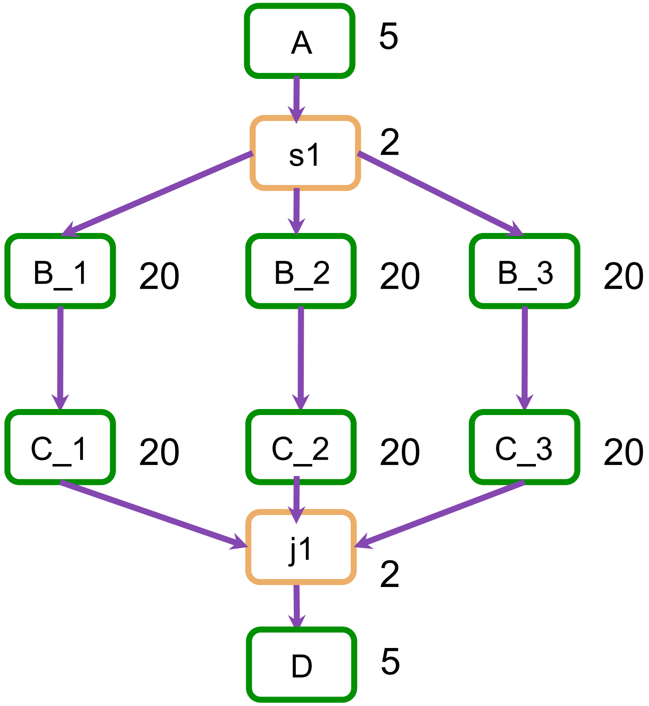


# Resolving Bottleneck Options



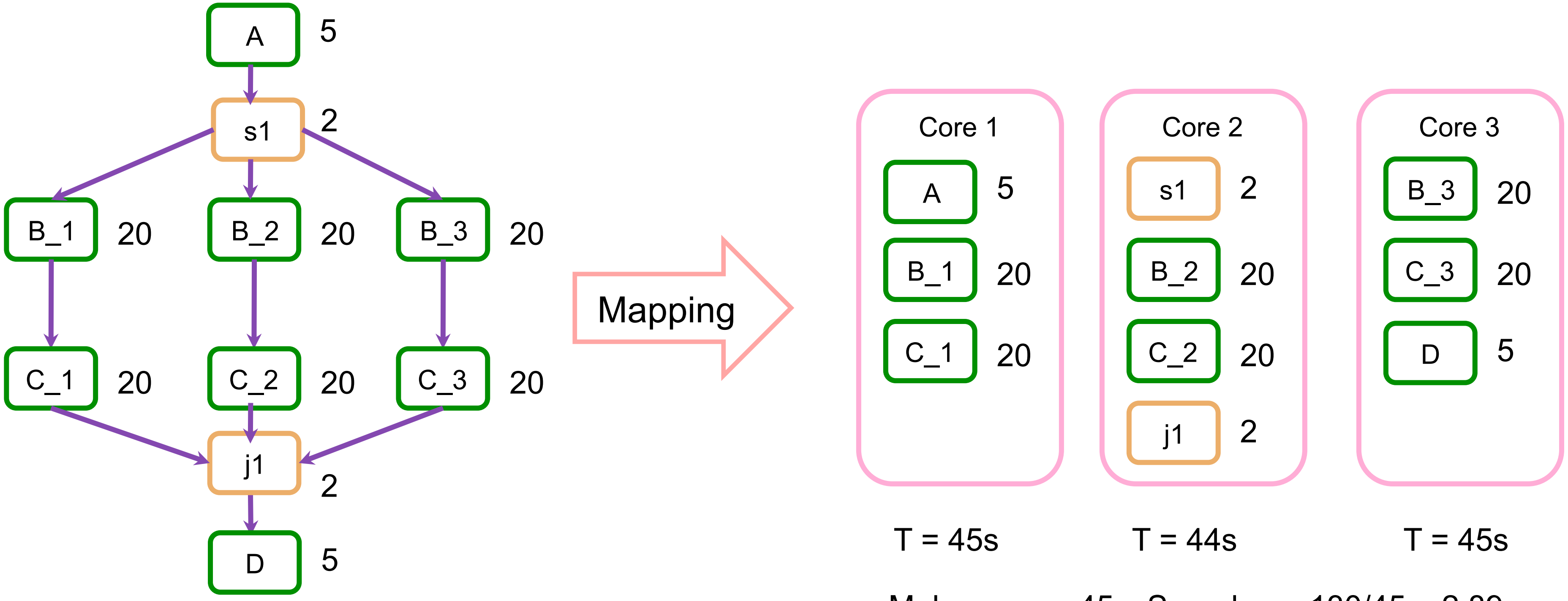


Hot actor duplication



Hot region duplication

# Region Duplication further Increases Performance

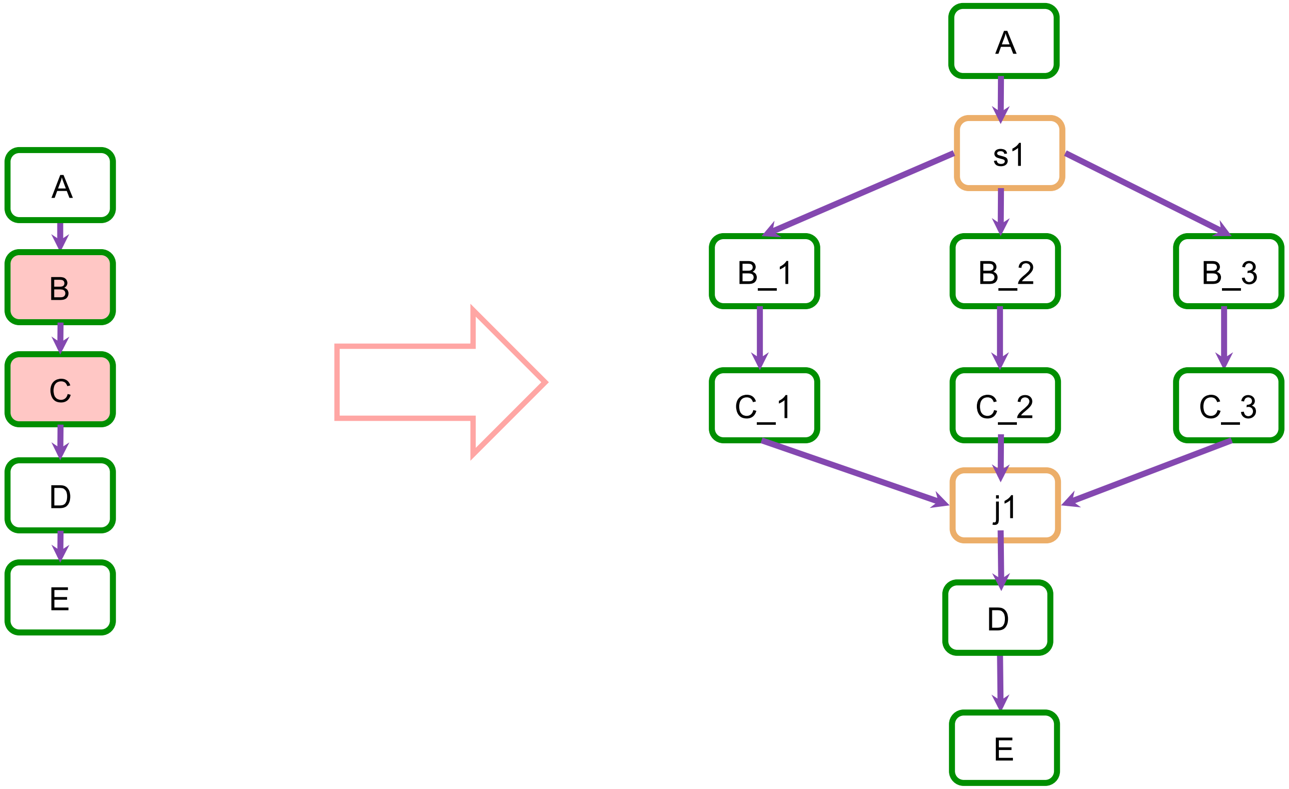


T = 45s T = 44s T = 45s

Make span = 45s, Speedup = 130/45 = 2.89

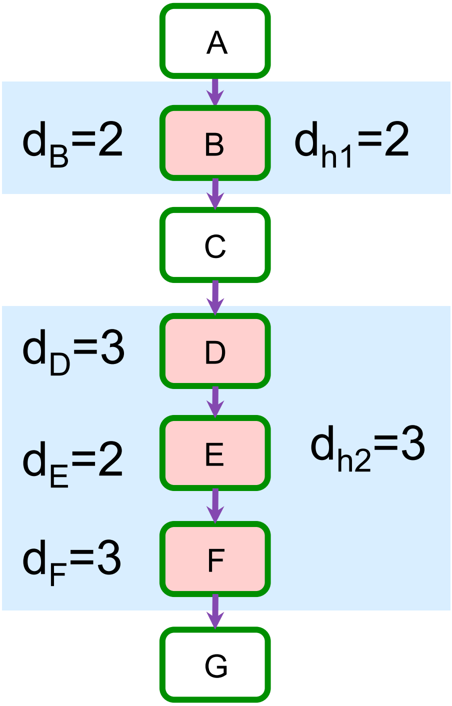
# Cascading Effect of Duplication

- Actors may become hot due to duplication of other actors



# Duplication Factor of an Actor and a Hot Region

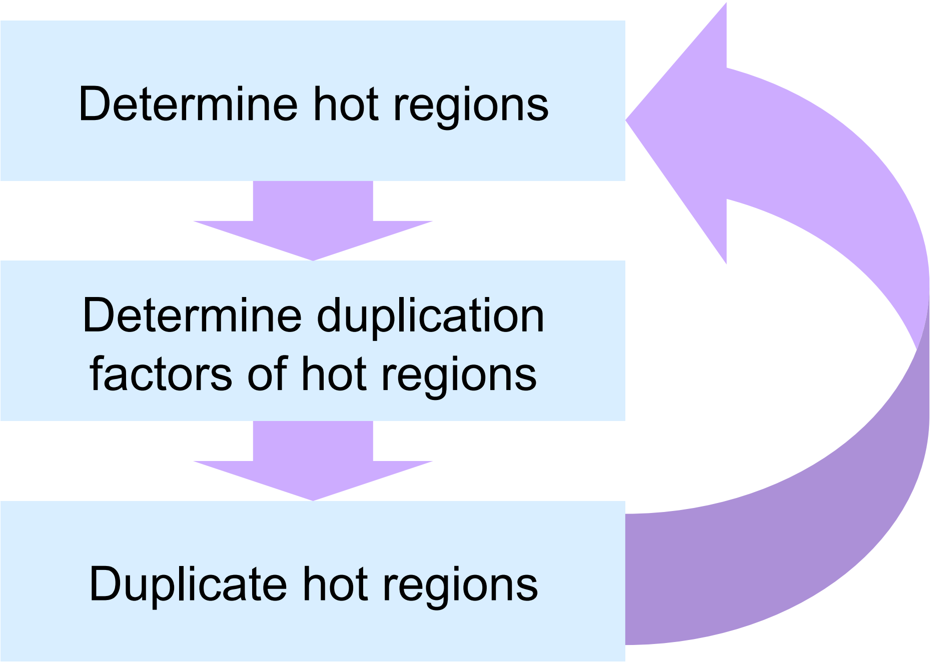
- The # of times the actor needs to be duplicated $d_i > 1$
- Maximum duplication factor of the actors of the hot region



# Heuristics to Resolve Bottlenecks

Optimal solution?

Experiment



Determine hot regions

Determine duplication factors of hot regions

Duplicate hot regions

# Summary

- A simple quantitative analysis to detect and eliminate bottlenecks
- We separate the bottleneck elimination from the actor allocation
- Heuristics to eliminate bottlenecks

# 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]

# Questions?