Performance benefit of single assignment languages for parallel execution

Paul B. Beskow (Cisco), Håkon K. Stensland (iAd Center),
Håvard Espeland (LABO Mixed Reality),
Preben Olsen, **Carsten Griwodz**, Pål Halvorsen
(Simula Research Lab)

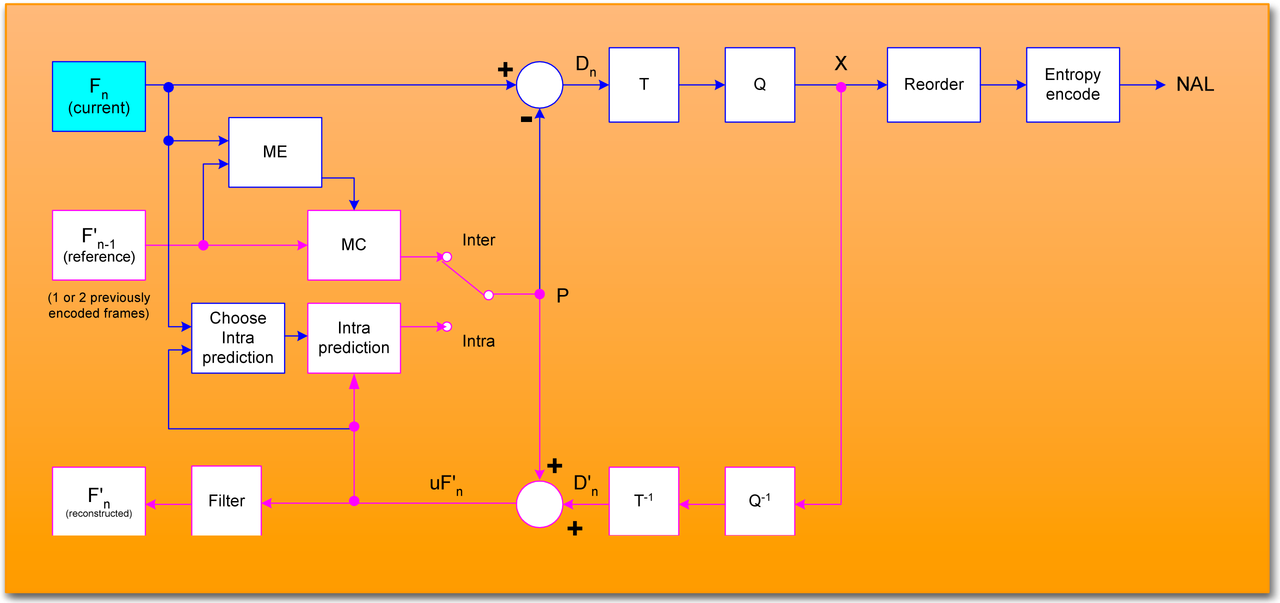
Parallel Processing for Multimedia Workloads

Multimedia workloads

- deadline-driven
- cyclic

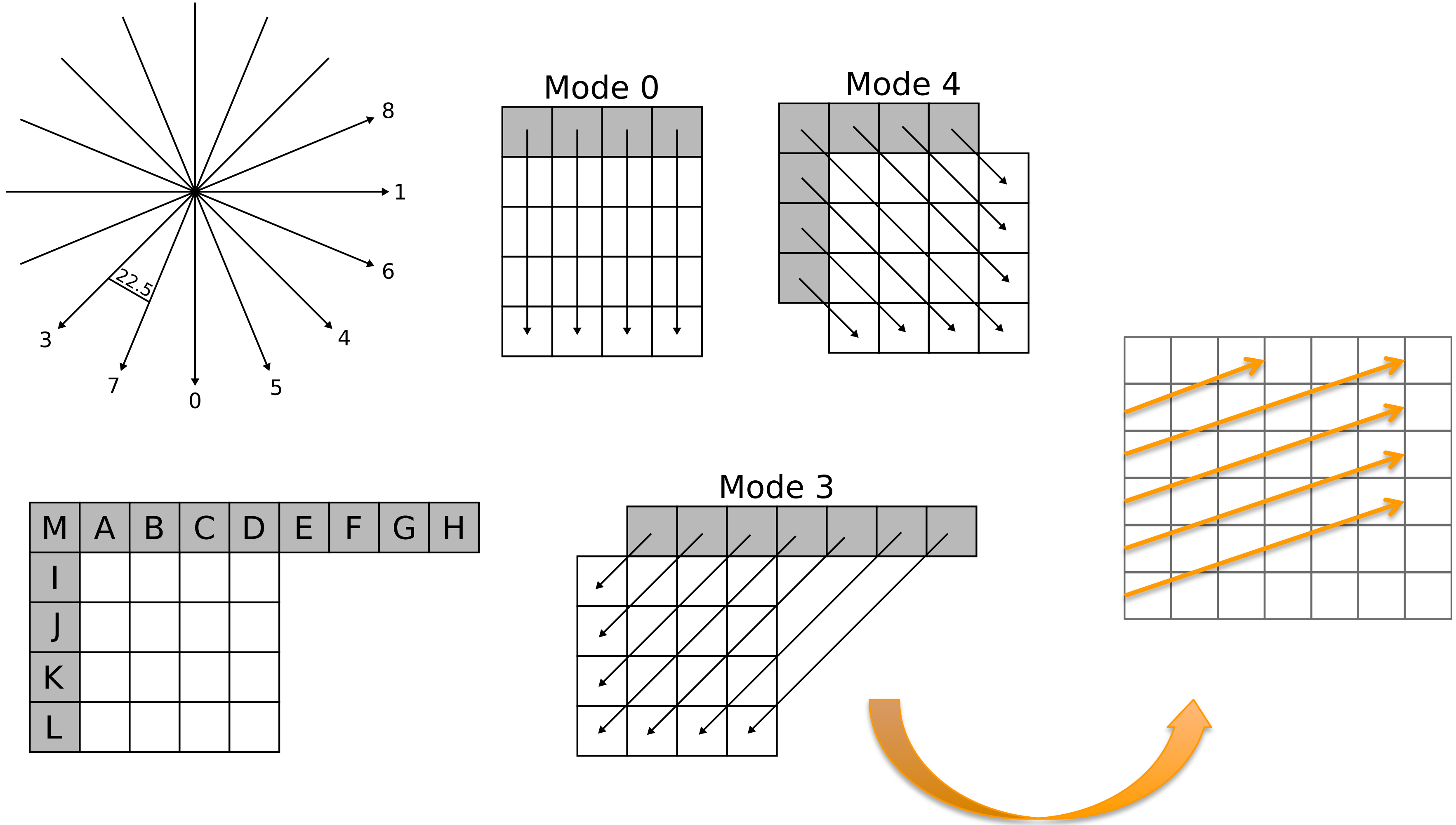
Multimedia algorithms

- long-range dependencies
- high parallelization potential

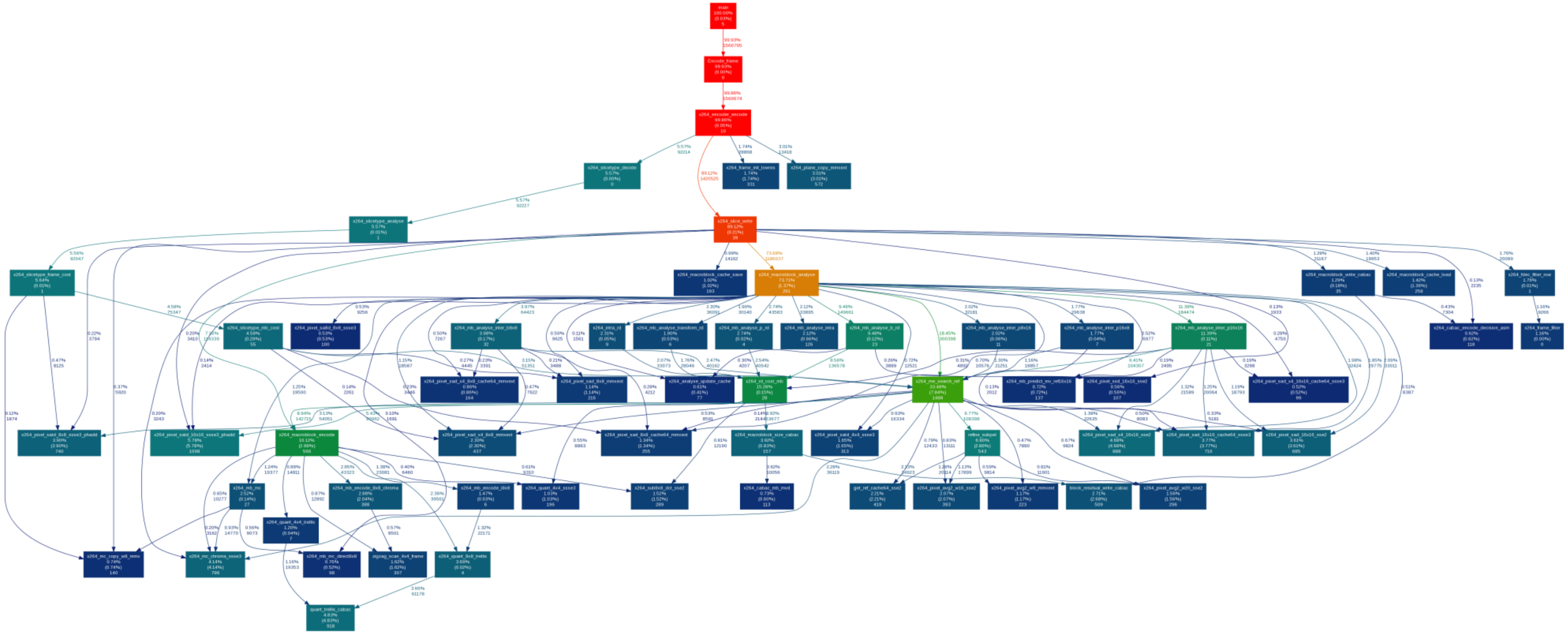


Parallel Processing for Multimedia Workloads

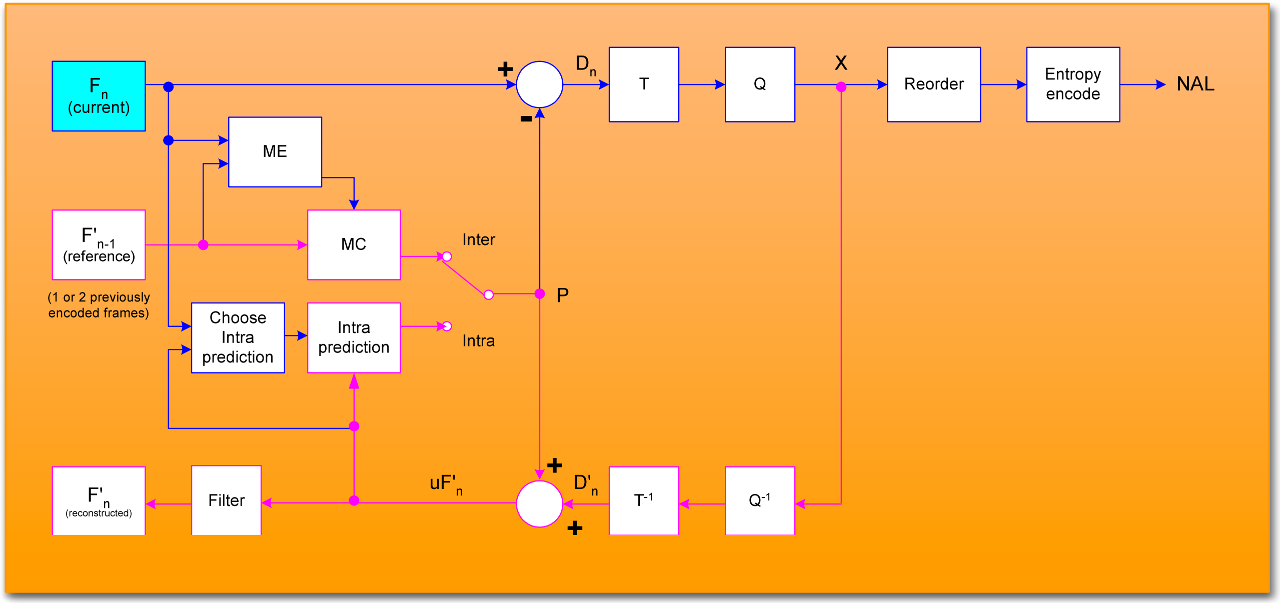
Even intra-module parallelization is not straight-forward



Parallel Processing for Multimedia Workloads
H.264 Encoder - x264 call graph



Typical Features of Multimedia Workloads



combination of several algorithms

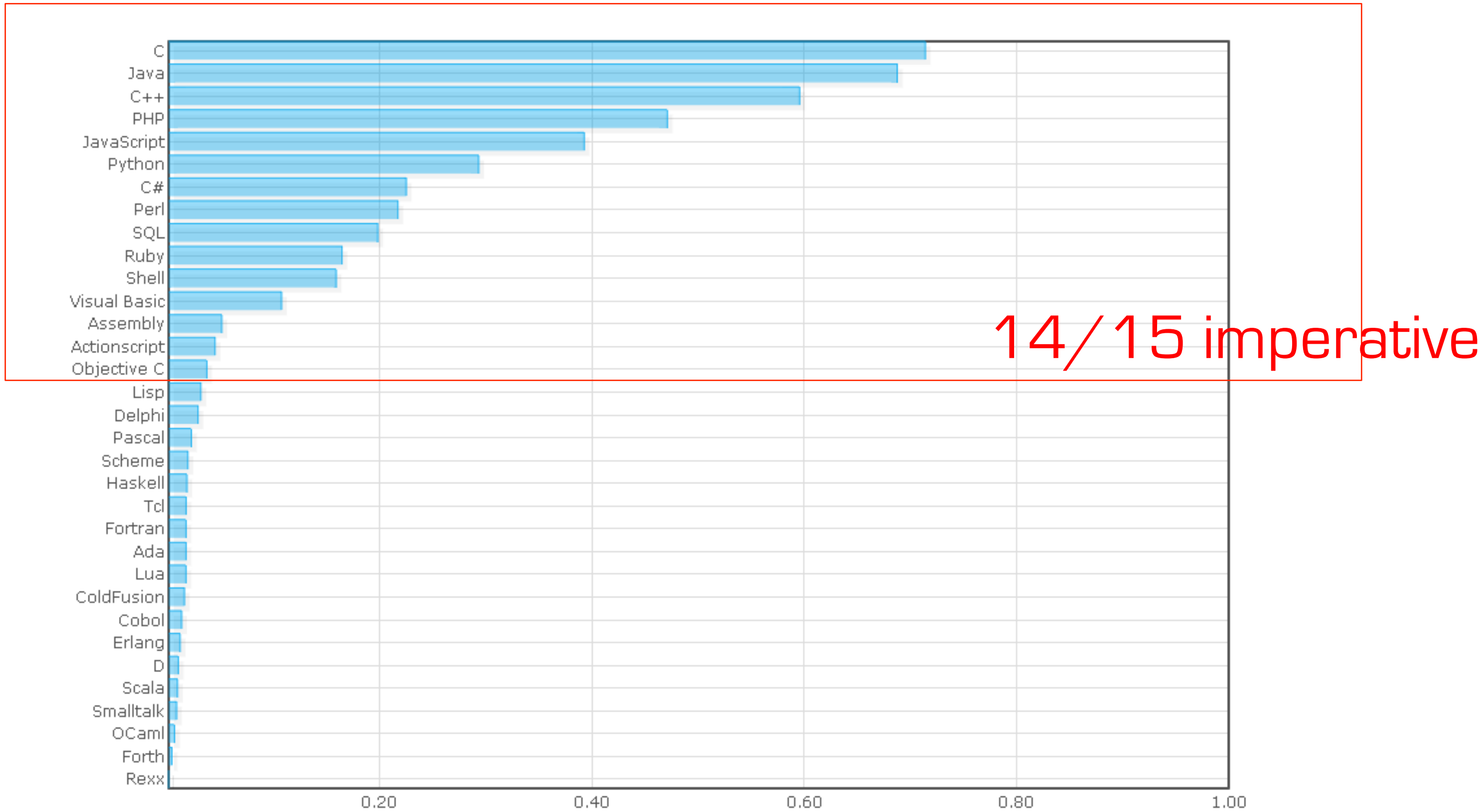
typically each specified at top level

connected by data transfer

long-range dependencies

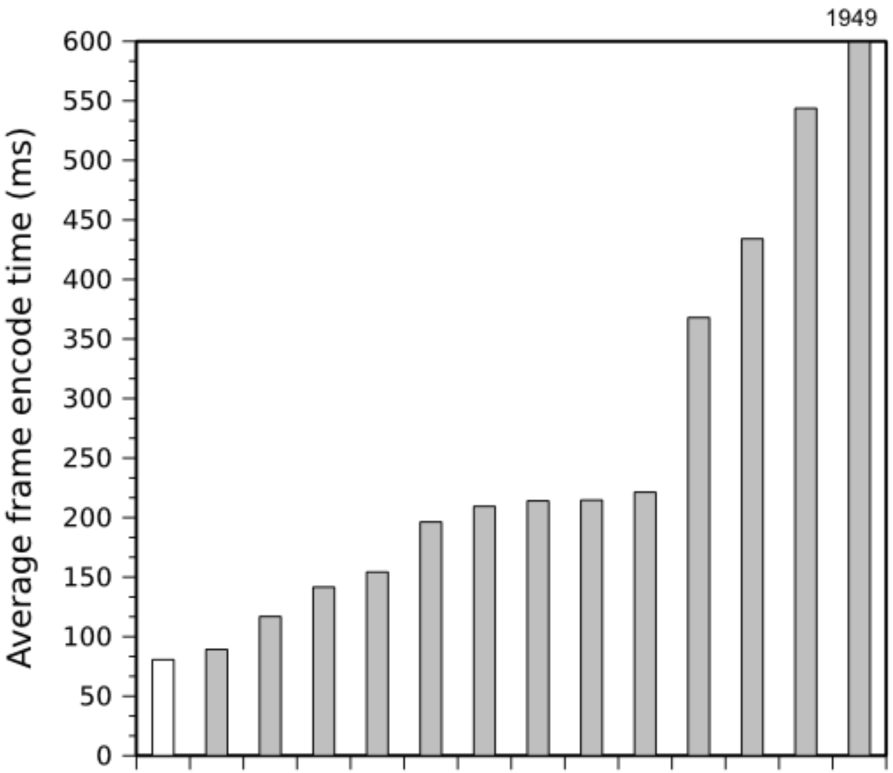
directed cyclic graph

Changing language concepts takes time

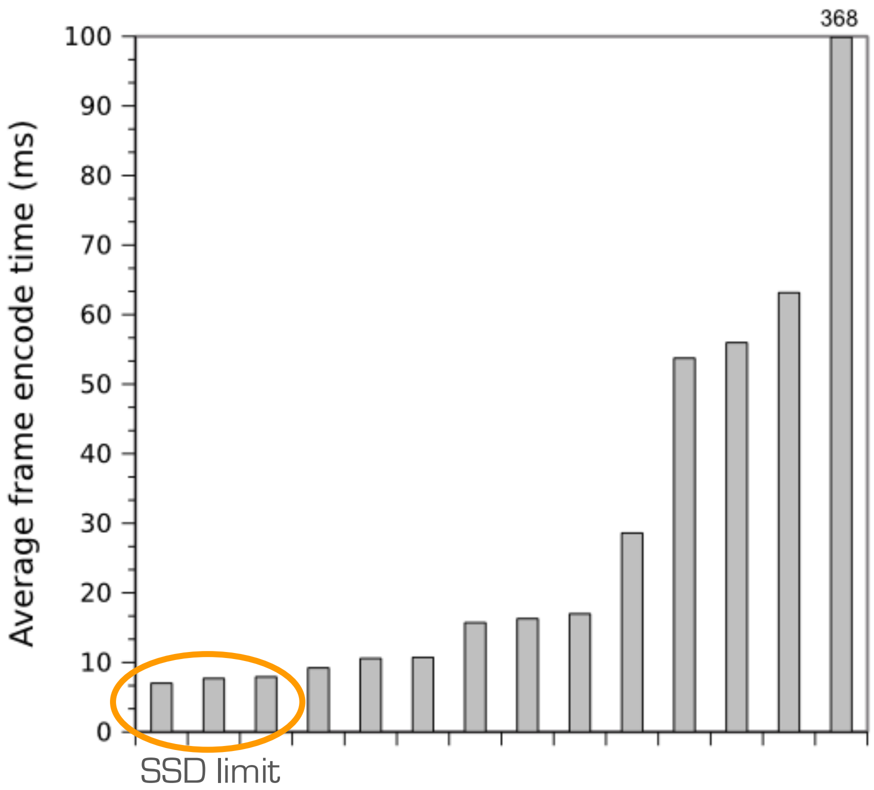


Primarily due to limitations in existing progamming models

Parallelization of a simple MPEG-like video encoder



Cell Broadband Engine



nVIDIA GeForce GPU

Design: Virtual Fields

Comparable to C++ multi dimensional arrays

Virtual

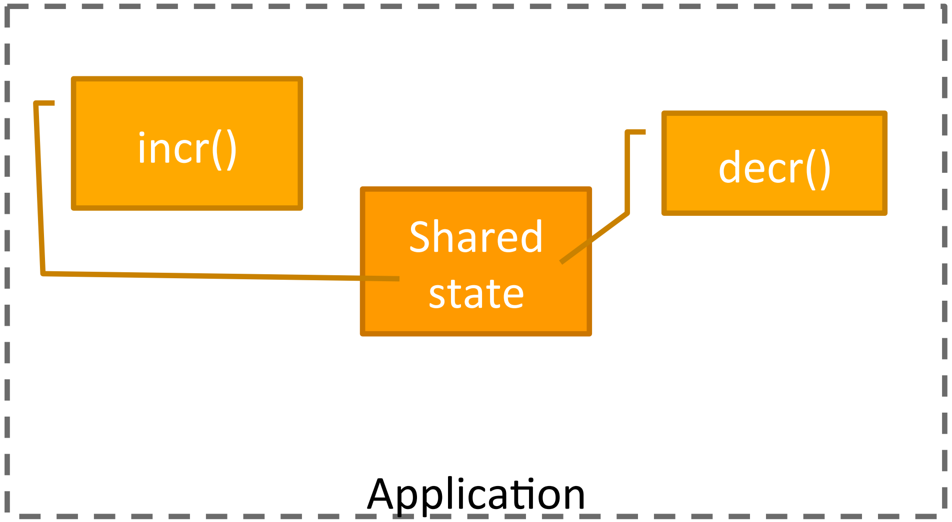
- Can be distributed
- Can be optimized out during compile or run-time

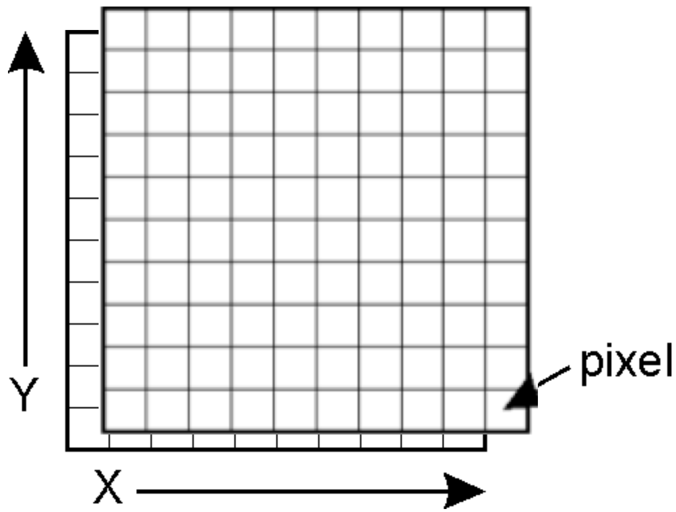
Write-once semantics

- Ensures deterministic execution

Aged fields

- Versioning
- Retains write-once semantics → Allows iterations (cycles)

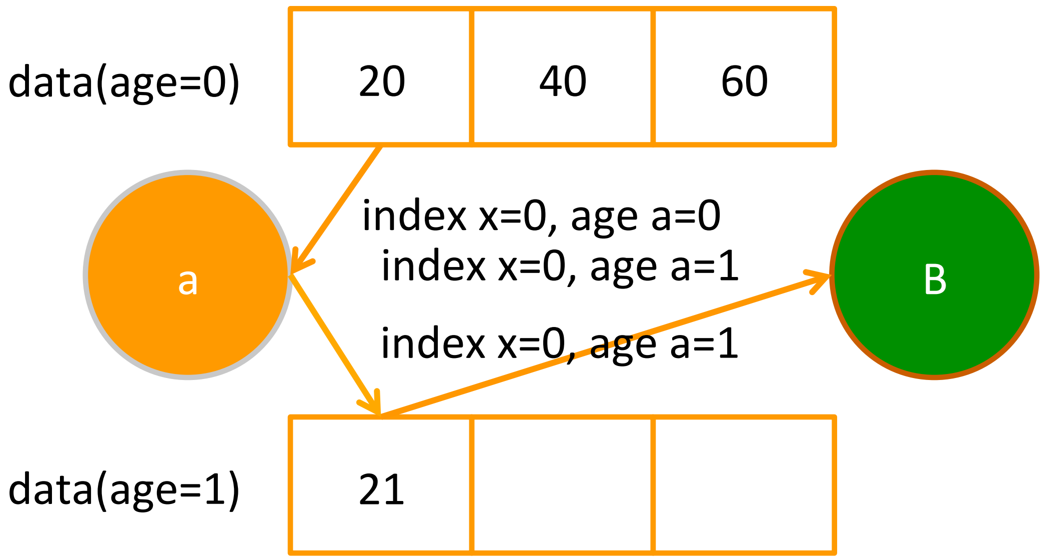




Design: Kernel

Embeds native C++ code
- Can use existing libraries or code bases

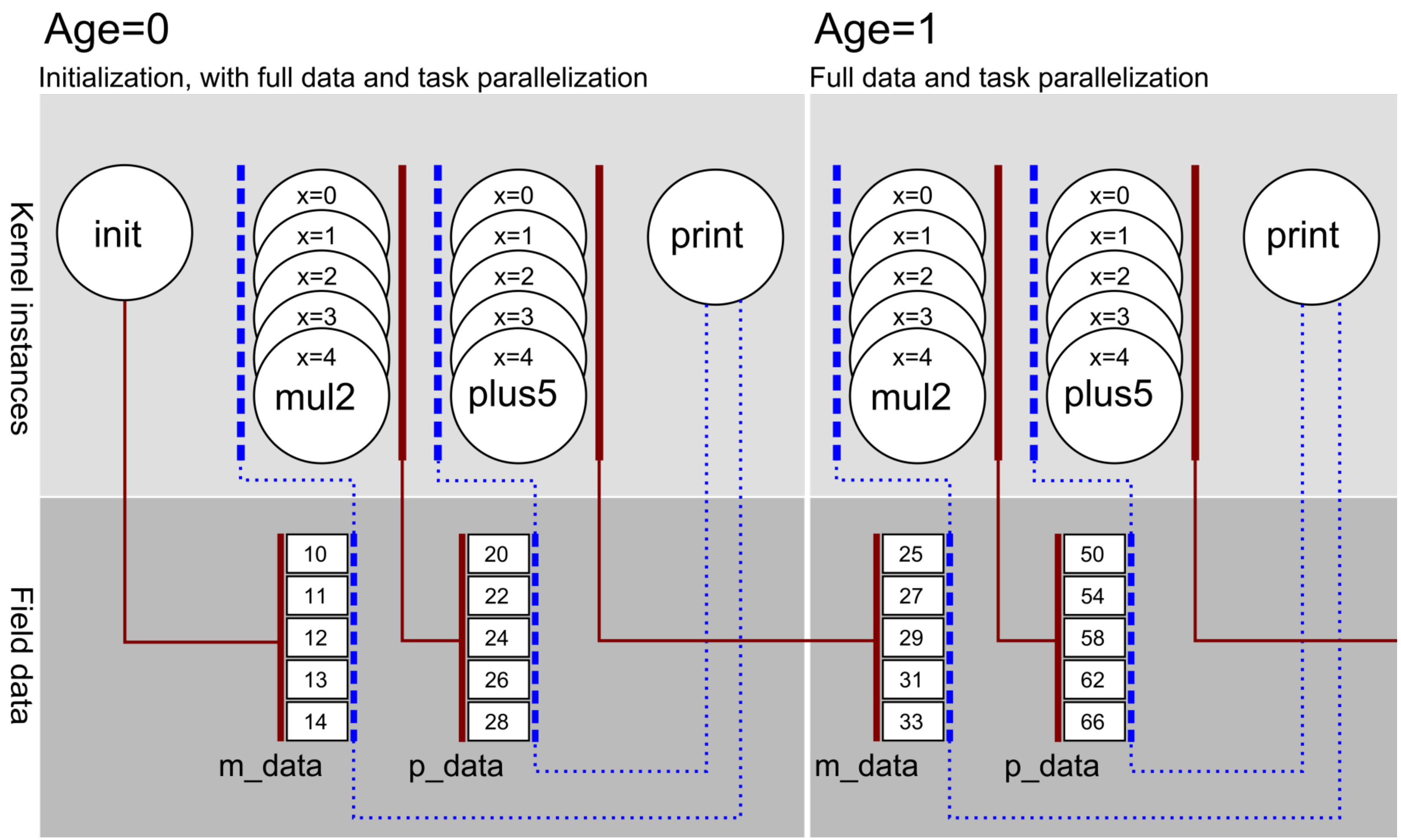
Dependencies expressed on *virtual fields*
- *Fetch* statements
  → *read* from a memory cell
- *Store* statements
  → *write* to a memory cell



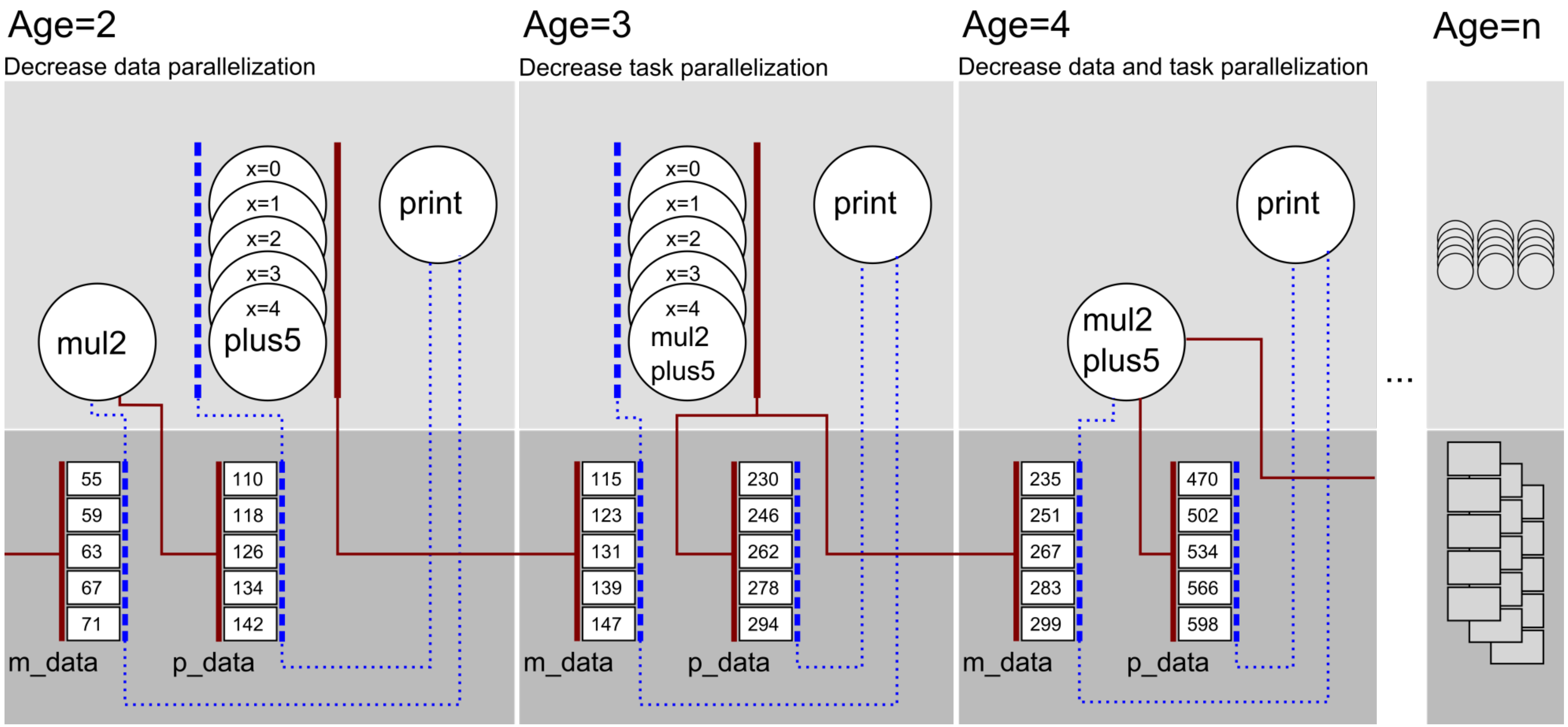

```
a:
  age a;
  index x;
  local int i;
  fetch i = data(a)[x];
  %{
      i += 1;
  %}
  store data(a+1)[x];
```

```
b:
  local int i;
  fetch i = data(1)[0];
```

Dynamic Dependency DAG



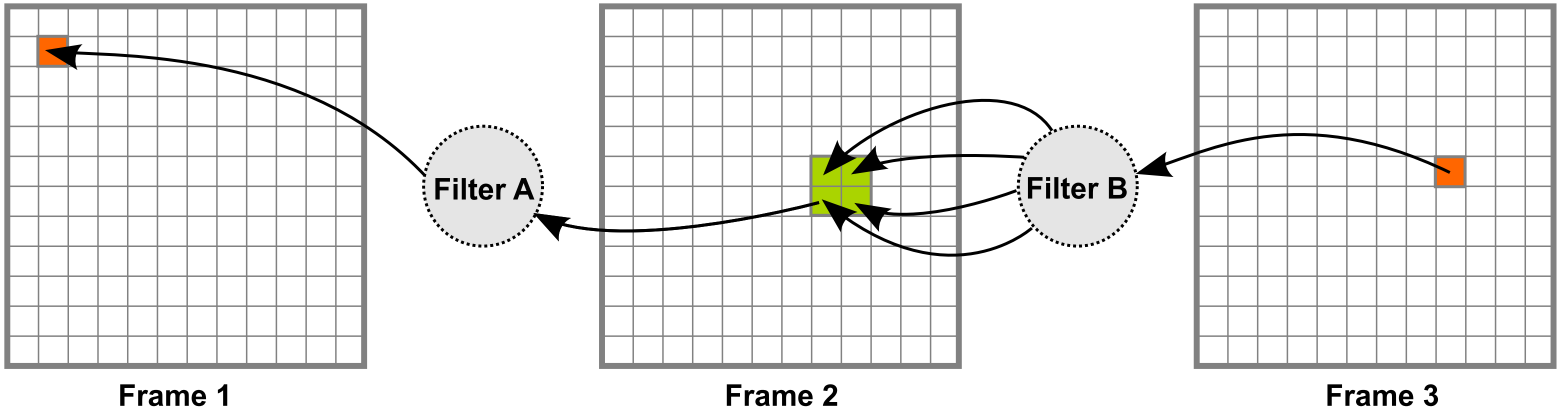
Granularity reduction



Dependencies

Straightforward implementations apply filters sequentially in order ➔ SEQ

Per-block, per-pixel, per-region pipelining may benefit from L1 caching; apply forward or backward ➔ BD



Re-use of pixel may call for explicit caching (write-back to memory) to avoid computation overhead ➔ BD-CACHED

Pipeline by architecture

Blur convolves the source frame with a Gaussian kernel to remove pixel noise.

Sobel X and Y are two filters that also convolve the input frame, but these filters apply the Sobel operator used in edge detection.

Sobel Magnitude calculates the approximate gradient magnitude using the results from Sobel X and Sobel Y.

Threshold unset every pixel value in a frame below or above a specified threshold.

Undistort removes barrel distortion in frames captured with wide-angle lenses. Uses bilinear interpolation to create a smooth end result.

Crop removes 20% of the source frame's height and width, e.g., a frame with a 1920x1080 resolution would be reduced to 1536x864.

Rotation rotates the source frame by a specified number of degrees. Bilinear interpolation is used to interpolate subpixel coordinates.

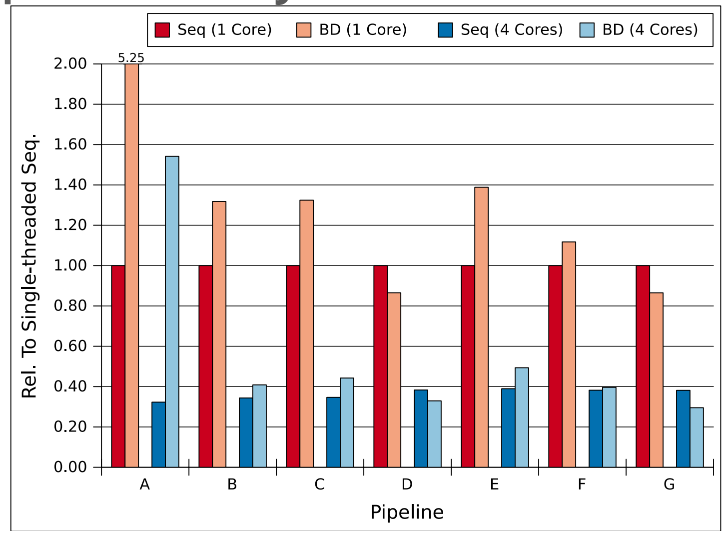
Discrete discretizes the source frame by reducing the number of color representations.

Binary creates a binary (two-colored) frame from the source. Every source pixel that is different from or above zero is set, and every source pixel that equals zero or less is unset.

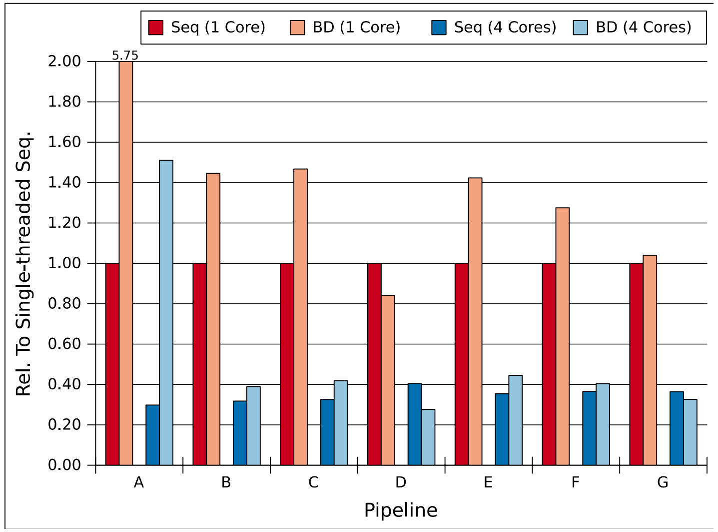
operations per pixel

Pipeline	Filter	Seq	BD	BD-CACHED
A	Blur	9.00	162.00	9.03
	Sobel X	9.00	9.00	9.00
	Sobel Y	9.00	9.00	9.00
	Sobel Magnitude	2.00	2.00	2.00
	Threshold	1.00	1.00	1.00
B	Undistort	4.00	10.24	2.57
	Rotate 6°	3.78	2.56	2.56
	Crop	1.00	1.00	1.00
C	Undistort	4.00	8.15	2.04
	Rotate 60°	2.59	2.04	2.04
	Crop	1.00	1.00	1.00
D	Discrete	1.00	1.00	1.00
	Threshold	1.00	1.00	1.00
	Binary	1.00	1.00	1.00
E	Threshold	1.00	3.19	0.80
	Binary	1.00	3.19	0.80
	Rotate 30°	3.19	3.19	3.19
F	Threshold	1.00	3.19	0.80
	Rotate 30°	3.19	3.19	3.19
	Binary	1.00	1.00	1.00
G	Rotate 30°	3.19	3.19	3.19
	Threshold	1.00	1.00	1.00
	Binary	1.00	1.00	1.00

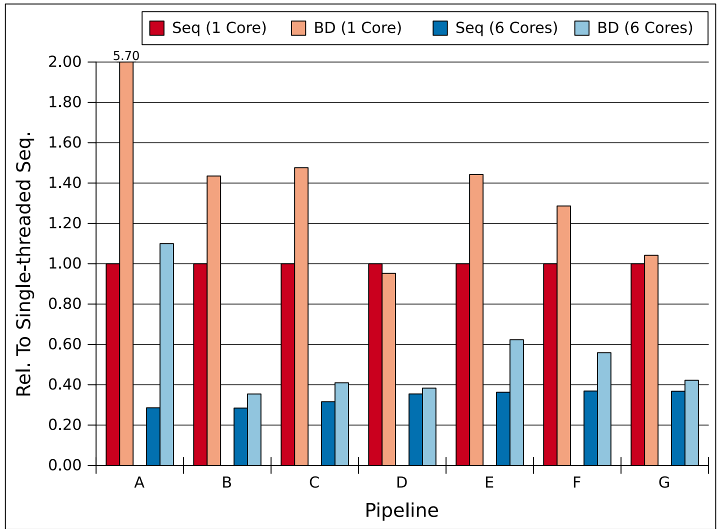
Pipeline by architecture



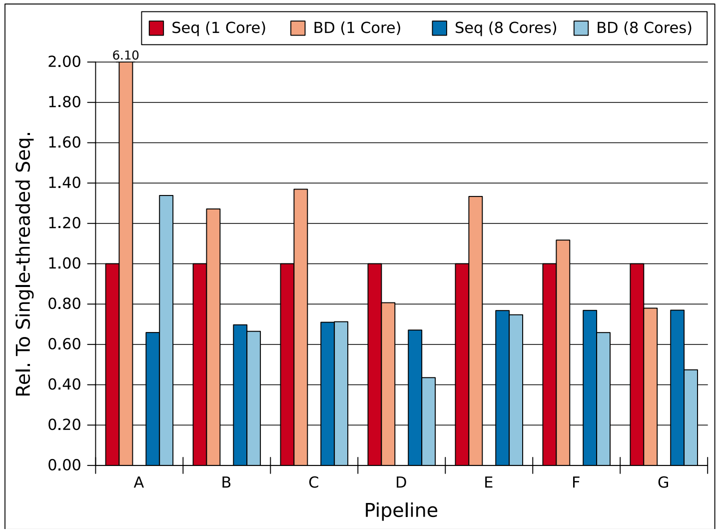
Nehalem



Sandy Bridge



Sandy Bridge-E



Bulldozer

Compiler support

Track dependencies through LLVM to code generation

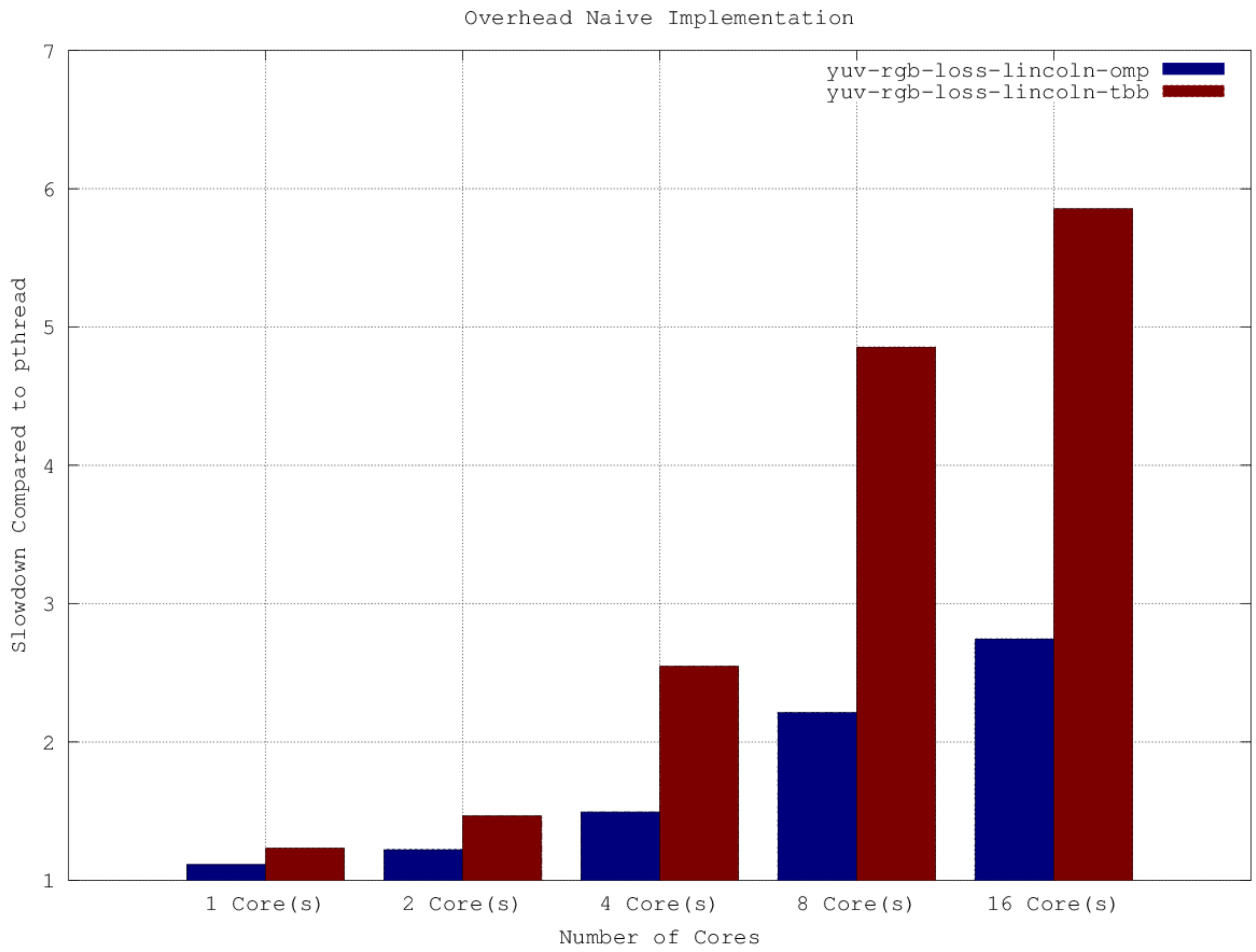
Merge kernel instances

- during code generation
  - e.g. subsequent one-to-one relationships are merged, but limit loop unrolling
- add code generation from intermediate representation at load time (compiler-rt)
  - partially generated IR
  - adapt loop size to thread pool size

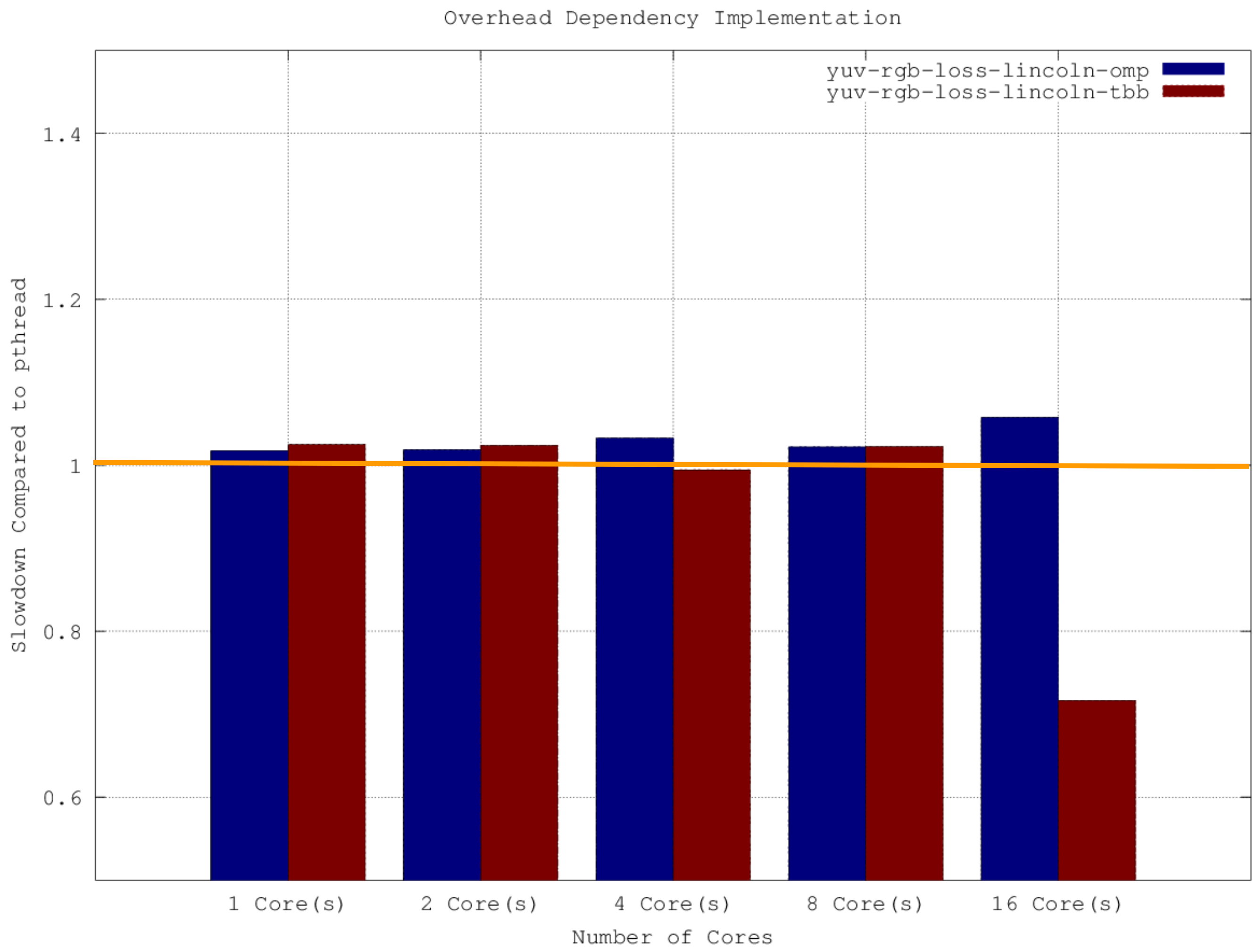
Run-time instantiation

- think of petri-nets to control instantiation
- but petri-net nodes are compile-time objects: decentralized, fast access

merging with p-thread run-time vs. TBB & OpenMP



p-thread run-time vs. TBB & OpenMP: all merging



Thank you!

Paul B. Beskow (Cisco), Håkon K. Stensland (iAd Center),
Håvard Espeland (LABO Mixed Reality),
Preben Olsen, **Carsten Griwodz**, Pål Halvorsen
(Simula Research Lab)