EE482S
Lecture 1
Stream Processor Architecture

April 4, 2002

William J. Dally
Computer Systems Laboratory
Stanford University
billd@csl.stanford.edu

Today's Class Meeting

- What is EE482C?
  - Material covered
  - Course format
  - Assignments
  - Scribing
- What is *Stream Processor Architecture?*
  - What problem is being solved
    - Performance scaling, power efficiency, bandwidth bottlenecks
  - What is a stream program?
  - What is a stream processor

What is EE482C?

- New course in EE482 sequence (advanced comp. arch.)
  - EE482A – superscalar architecture
  - EE482B – interconnection networks
  - EE482C – stream processor architecture
- Course format
  - Readings – typically one or two papers per class meeting
    - Read the paper before the meeting for which it is listed
    - E.g., read Rixner et al. before 4/9/02
  - Mix of lecture and discussion in class meetings
    - Be prepared to discuss each reading
  - Two programming assignments
    - One in StreamC/KernelC, one in Brook
  - Class project
    - Original research on stream architecture or programming

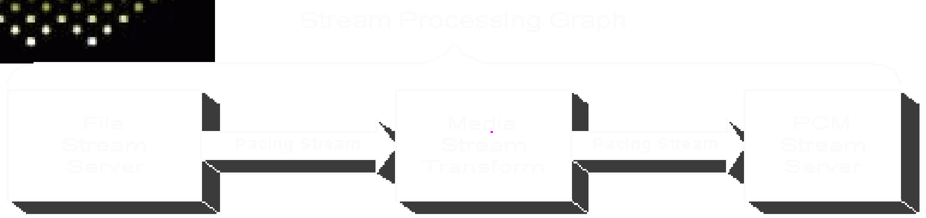
Where to find more information

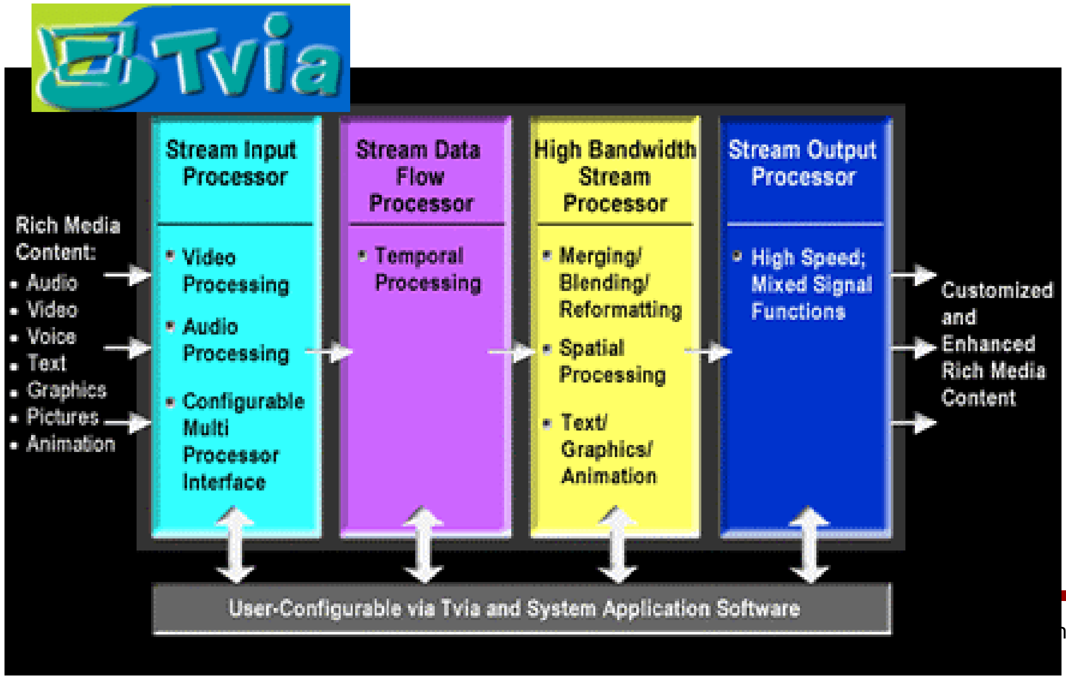
- Course policy sheet
- Web page http://cva.stanford.edu/ee482s
- Class schedule
  - Subject to change

What is a Stream Processor?

- A stream program is a computation organized as streams of records operated on by computation kernels.
- A stream processor is optimized to exploit the locality and concurrency of stream programs
- More later

Stream Processing is becoming pervasive




Tvia
Rich Media Content:
• Audio
• Video
• Voice
• Text
• Graphics
• Pictures
• Animation
Stream Input Processor
• Video Processing
• Audio Processing
• Configurable Multi Processor Interface
Stream Data Flow Processor
• Temporal Processing
High Bandwidth Stream Processor
• Merging/ Blending/ Reformatting
• Spatial Processing
• Text/ Graphics/ Animation
Stream Output Processor
• High Speed; Mixed Signal Functions
Customized and Enhanced Rich Media Content
User-Configurable via Tvia and System Application Software

Insights

Peter Huber, 01.07.02

A new type of computing--"stream computing"--has emerged to handle the back end of sonar, radar, X-ray sources and certain broadband applications such as voice-over Internet and digital TV.

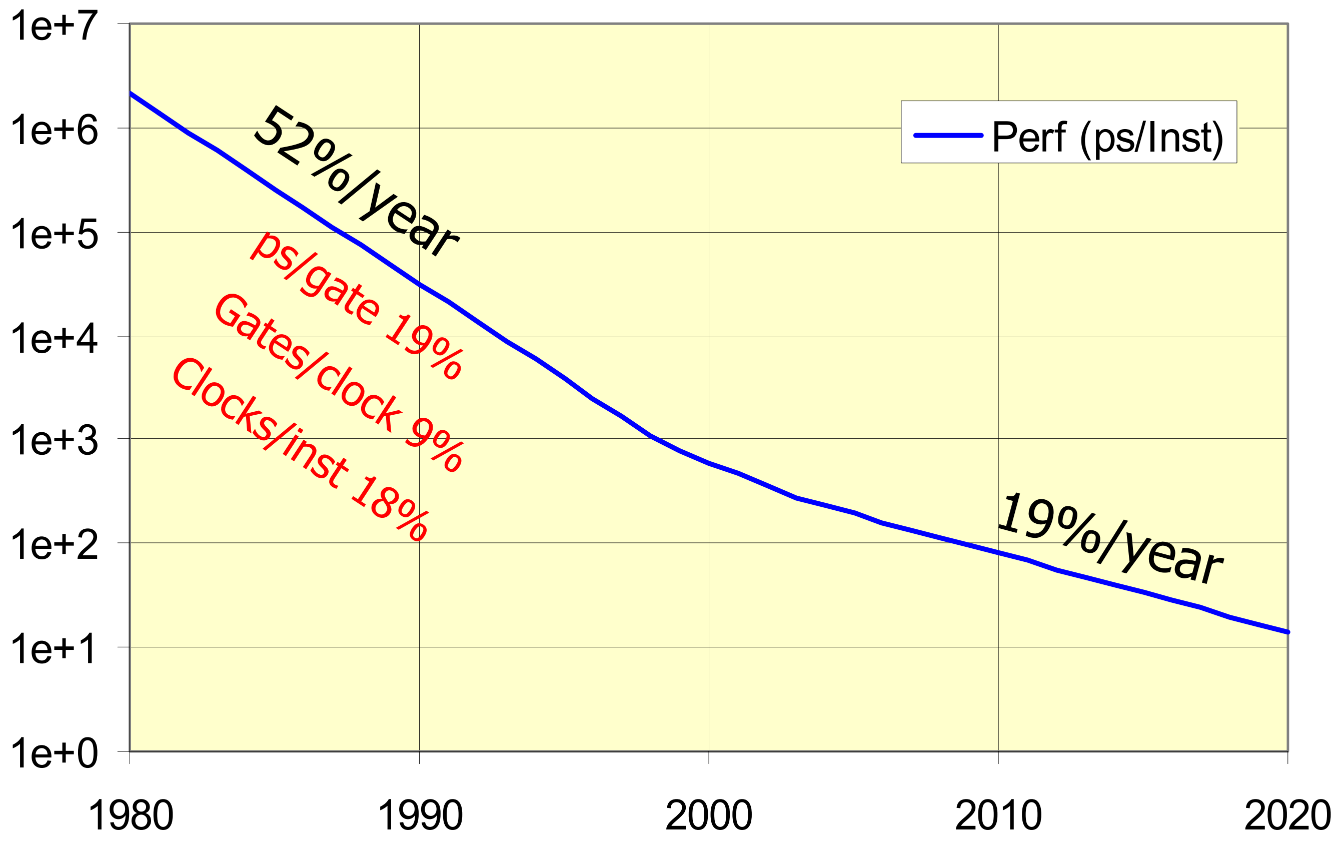
Motivation – Why do we need a stream processor?

- Application demand
- Power
- Bandwidth
- Performance Scaling

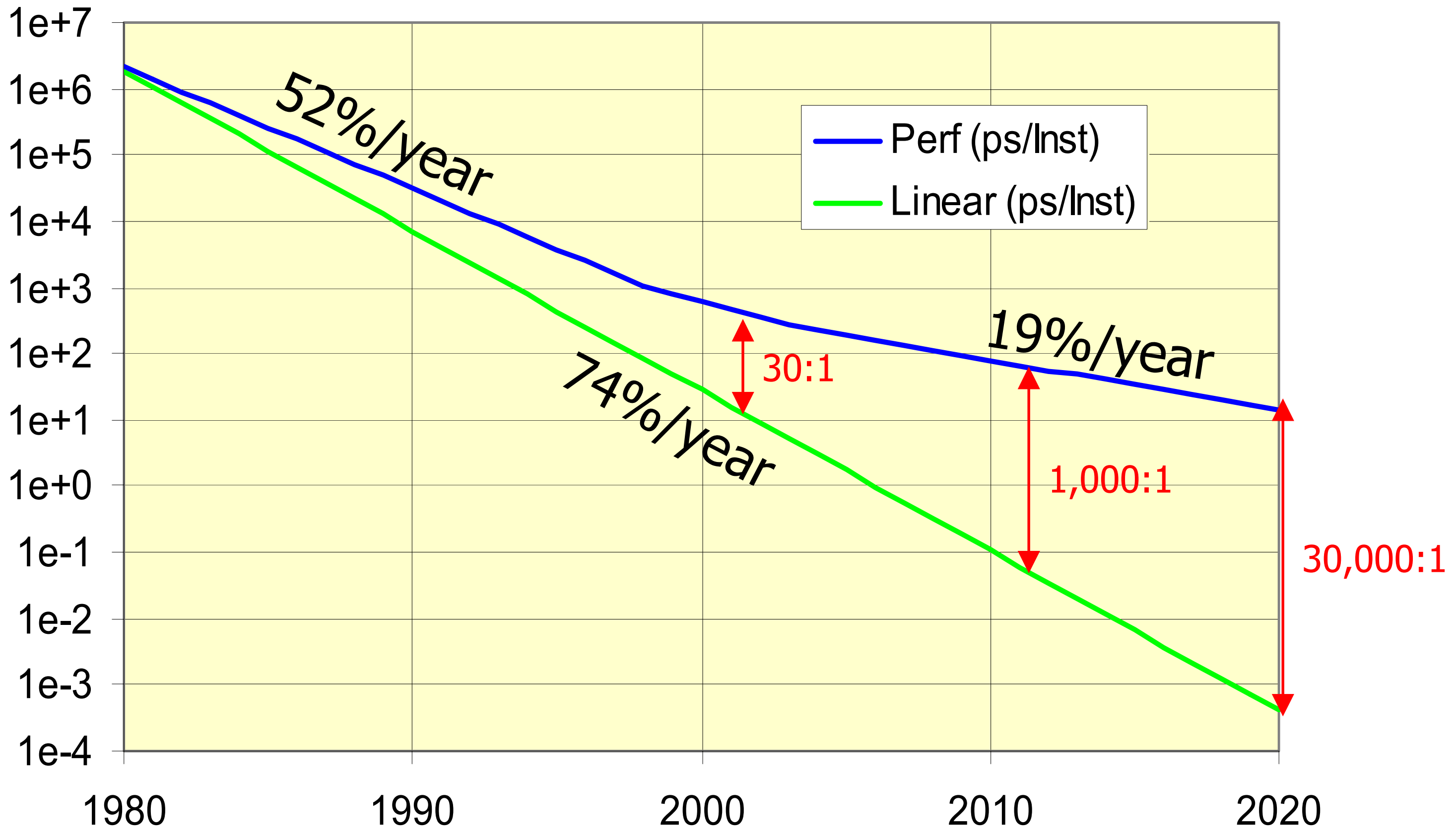
Application Pull

- Emerging media applications demand 10s of GOPS to TOPs with low power.
  - Video compression
  - Image understanding
  - Signal processing
  - Graphics
- Scientific applications also need TOPs of performance with reasonable cost

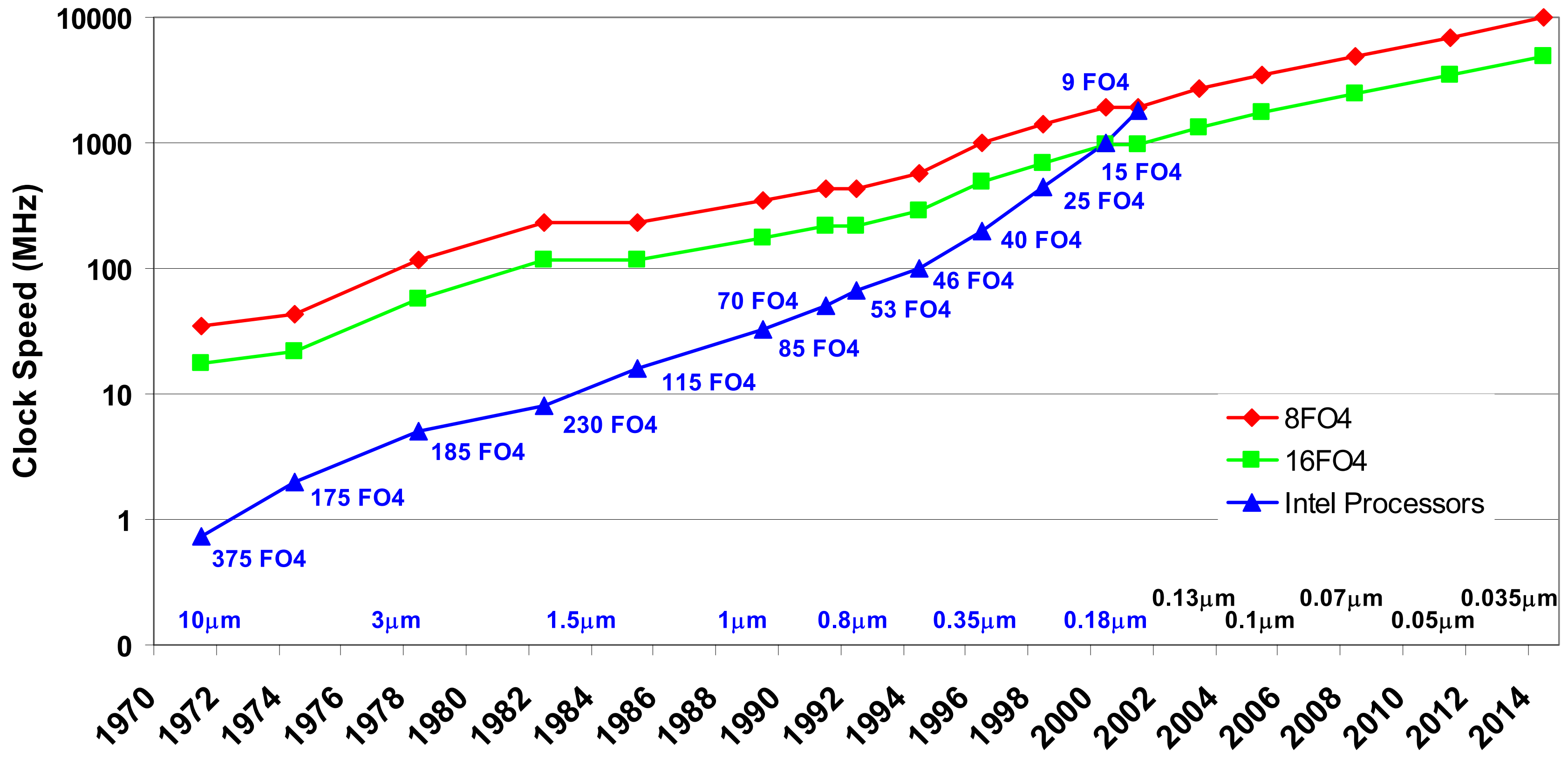
Conventional Processors No Longer Scale Performance by 50% each year



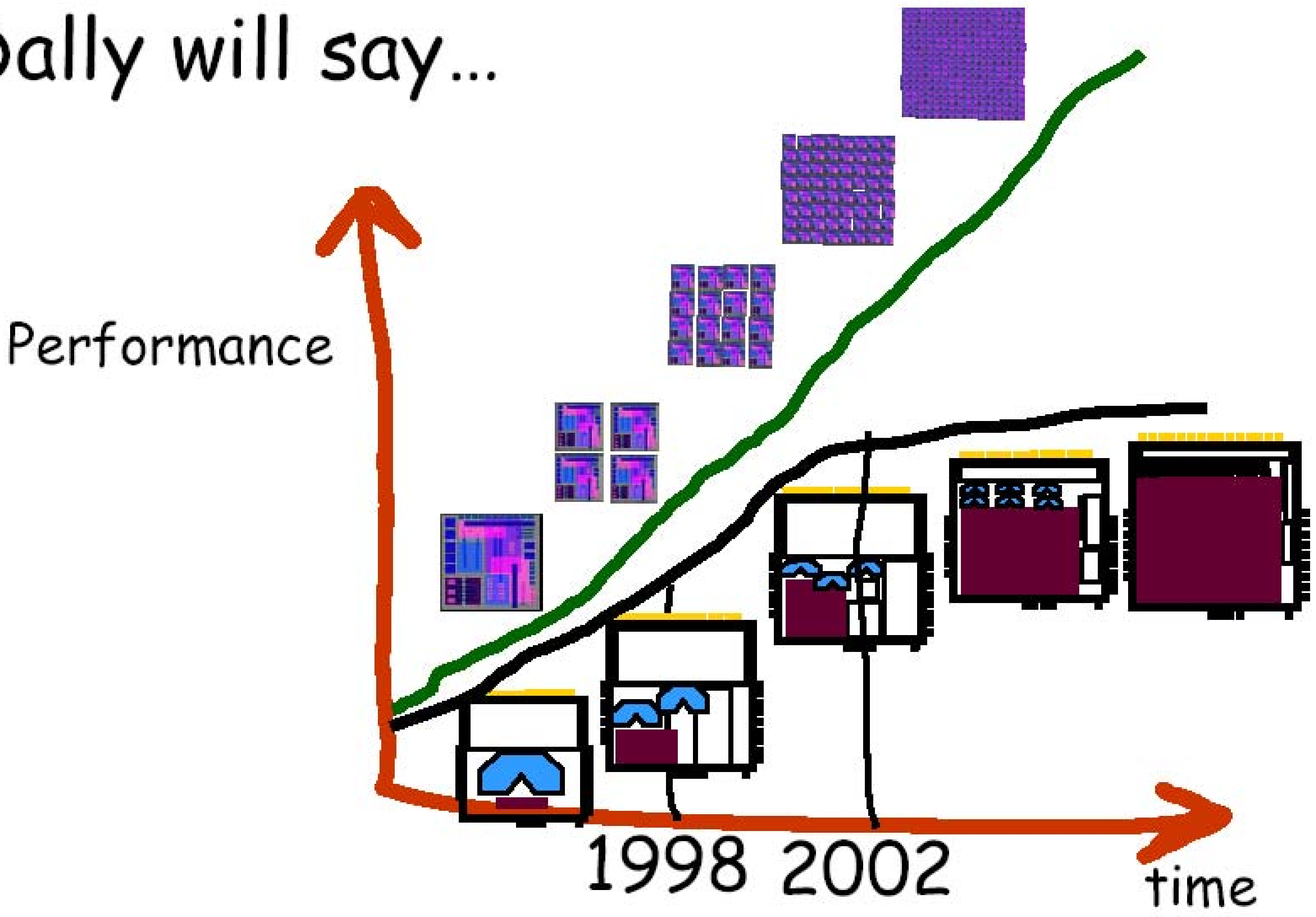
Future potential of novel architecture is large (1000 vs 30)



Clock Scaling: Historical and Projected



Dally will say…

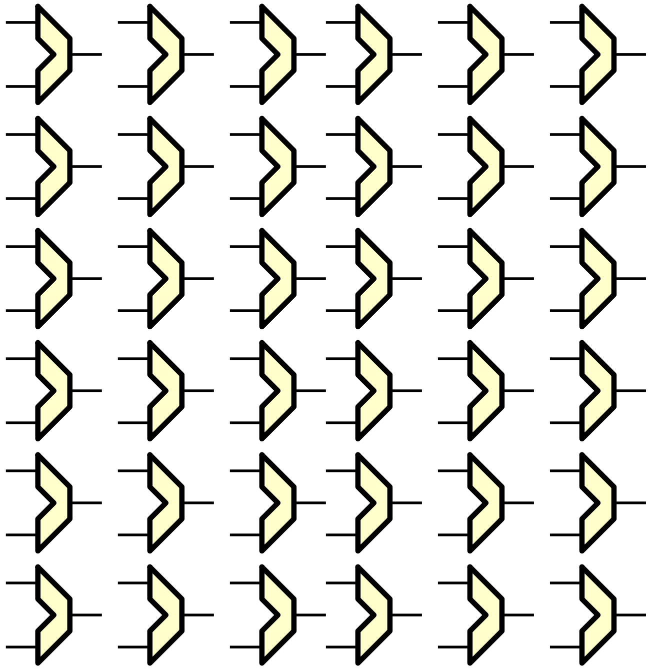


Anant Agarwal (MIT) Panel at HPCA '02

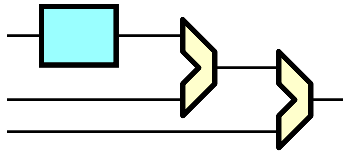
Pentium III vs. Pentium IV

	Pentium III	Pentium IV
Technology	180nm	180nm
Die Size	106mm^2	217mm^2
Transistor Count	24 million	42 million
# Grids	10^8	2×10^8
Pipeline Stages	10	20
Clock Rate	1GHz (15 FO4)	1.5GHz (10.4 FO4)
L1 D$ Capacity	16KBytes	12KBytes
SpecInt2000	454	524
SpecInt/MHz	**0.45**	**0.35**

Why do Special-Purpose Processors Perform Well?

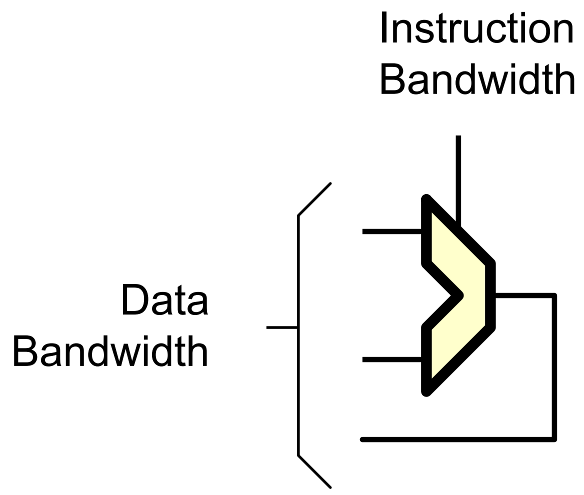


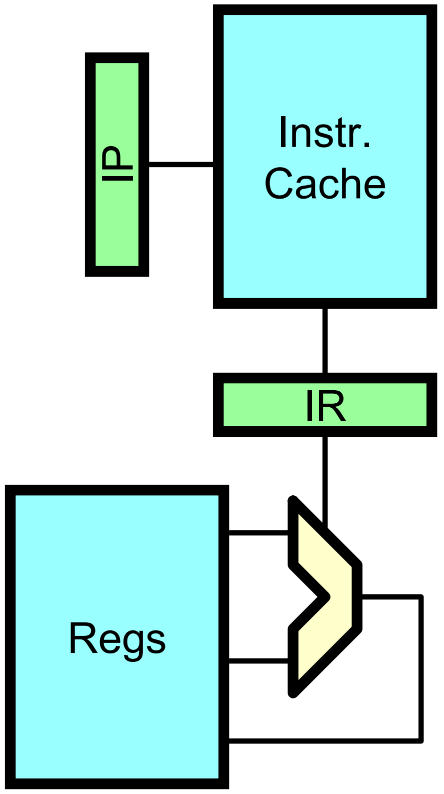
Lots (100s) of ALUs



Fed by dedicated wires/memories

Care and Feeding of ALUs





'Feeding' Structure Dwarfs ALU

Stream Programs make communication explicit

- This reduces energy and delay

Energy is a matter of distance (interconnect)

Operation	Energy (0.13um)	(0.05um)
32b ALU Operation	5pJ	0.3pJ
32b Register Read	10pJ	0.6pJ
Read 32b from 8KB RAM	50pJ	3pJ
Transfer 32b across chip (10mm)	100pJ	17pJ
Execute a uP instruction (SB-1)	1.1nJ	130pJ
Transfer 32b off chip (2.5G CML)	1.3nJ	400pJ
Transfer 32b off chip (200M HSTL)	1.9nJ	1.9nJ

300: 20: 1 off-chip to global to local ratio in 2002
1300: 56: 1 in 2010

Interconnect dominates delay

Operation	**Delay** (0.13um)	(0.05um)
32b ALU Operation	650ps	250ps
32b Register Read	325ps	125ps
Read 32b from 8KB RAM	780ps	300ps
Transfer 32b across chip (10mm)	1400ps	2300ps
Transfer 32b across chip (20mm)	2800ps	4600ps

2:1 global on-chip comm to operation delay
9:1 in 2010

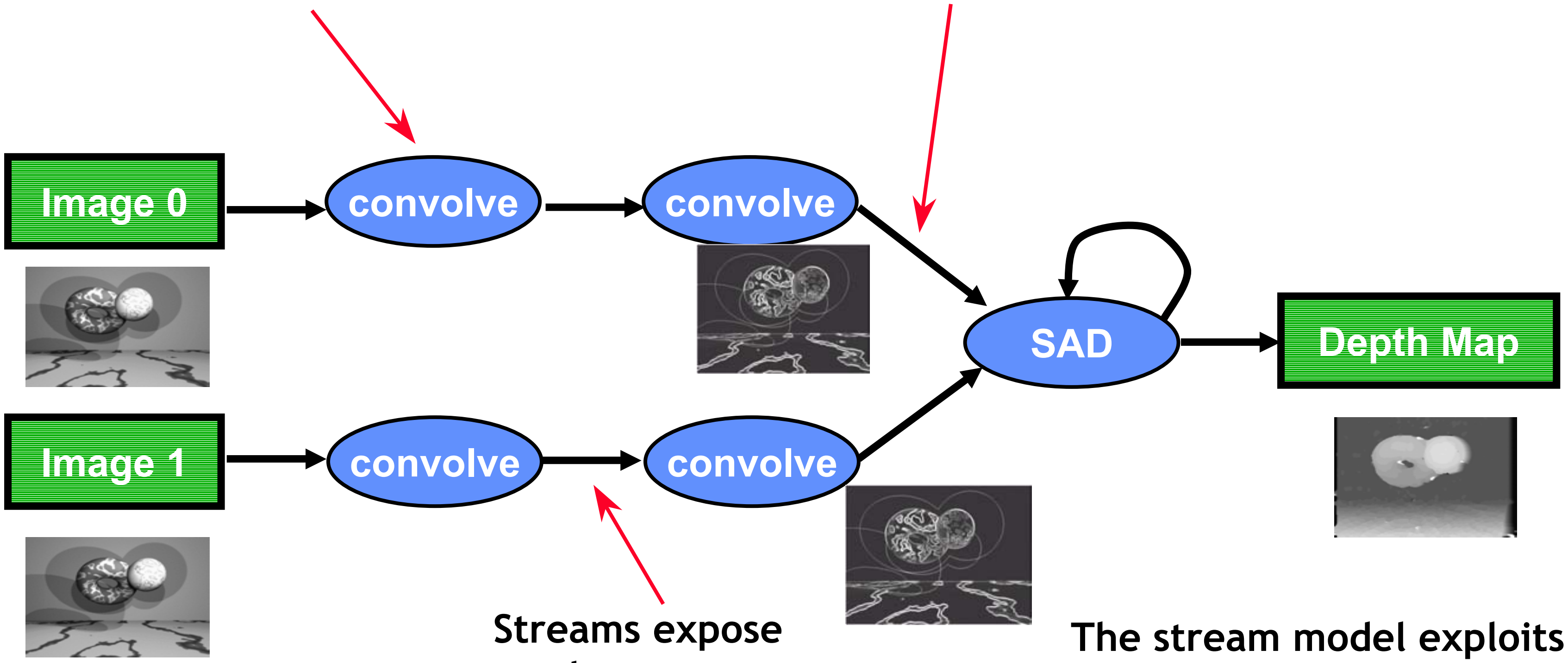
What is a Stream Program?

- A program organized as streams of records flowing through kernels
- Example, stereo depth extraction

Stereo Depth Extraction Stream Program

Kernels exploit both instruction (ILP) and data (SIMD) level parallelism.

Kernels can be partitioned across chips to exploit task parallelism.



Streams expose producer-consumer locality.

The stream model exploits parallelism without the complexity of traditional parallel programming.

Why Organize an Application This Way?

- Expose ***parallelism*** at three levels
  - ILP within kernels
  - DLP across stream elements
  - TLP across sub-streams and across kernels
  - Keeps ‘easy’ parallelism easy
- Expose ***locality*** in two ways
  - Within a kernel – kernel locality
  - Between kernels – producer-consumer locality
  - This locality can be exploited *independent* of spatial or temporal locality
  - Put another way, stream programs make *communication explicit*

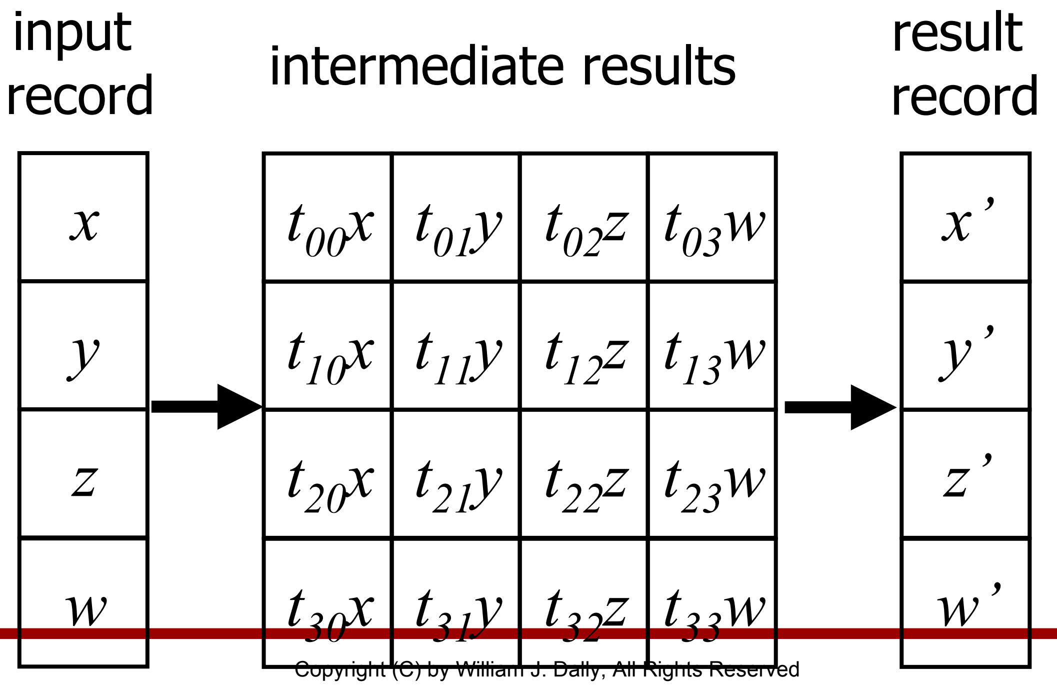
Streams expose *Kernel Locality* missed by **Vectors**

- Streams
  - Traverse operations first
    - All operations for one record, then next record
    - Smaller working set of temporary values
  - Store and access whole records as a unit
    - Spatial locality of memory references
      - e.g., get contiguous record on gather/scatter

- Vectors
  - Traverse records first
    - All records for one operation, then next operation
    - Large set of temporary values
  - Group like-elements of records into vectors
    - Read one word of each record at a time
      - No locality on gather/scatter

Example – Vertex Transform

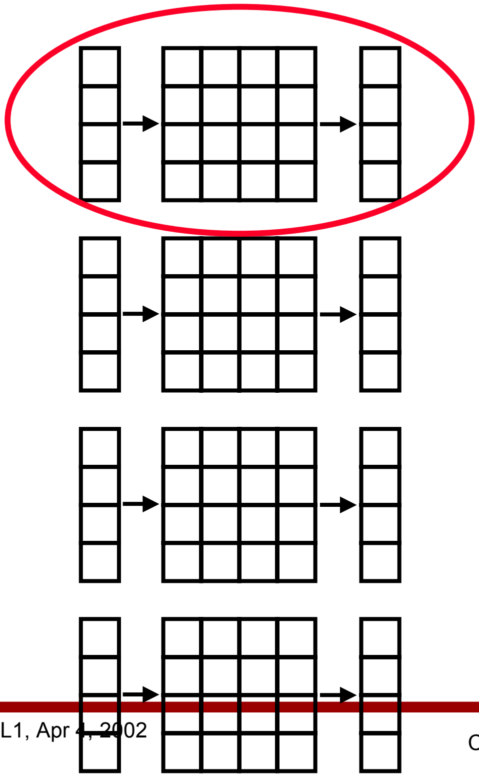
$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = T \times \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

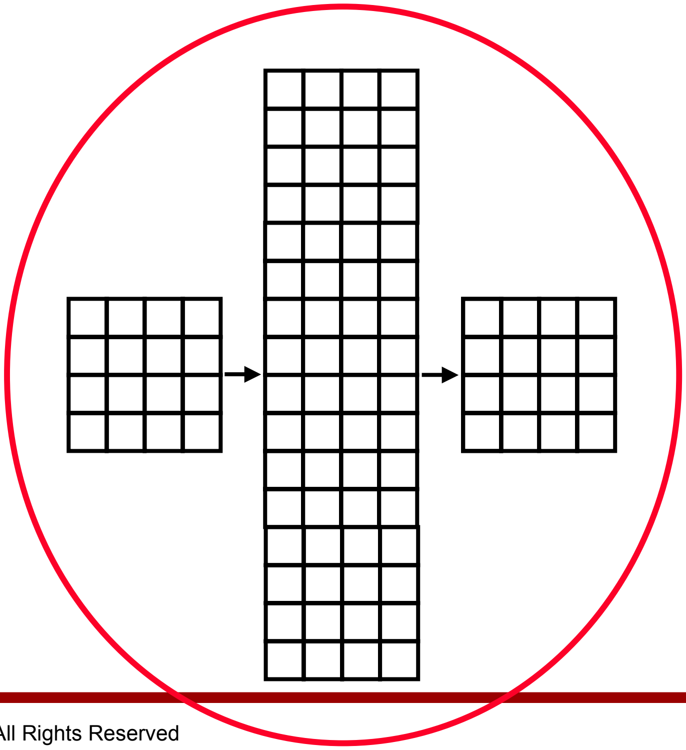

input record
intermediate results
result record
x
y
z
w
$t_{00}x$ $t_{01}y$ $t_{02}z$ $t_{03}w$
$t_{10}x$ $t_{11}y$ $t_{12}z$ $t_{13}w$
$t_{20}x$ $t_{21}y$ $t_{22}z$ $t_{23}w$
$t_{30}x$ $t_{31}y$ $t_{32}z$ $t_{33}w$
x'
y'
z'
w'

Vertex transform with Streams vs. Vectors

- Small set of intermediate results
  - enable small and fast LRFs

- Large working set of intermediates
  - VL times larger (e.g. 64x)
  - Must use a large, slow global VRF

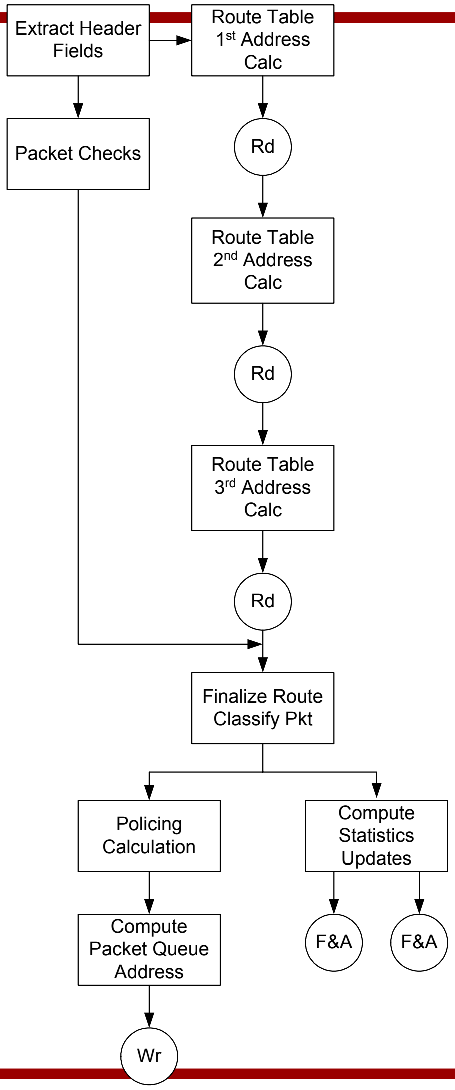




What class of applications can be written as stream programs?

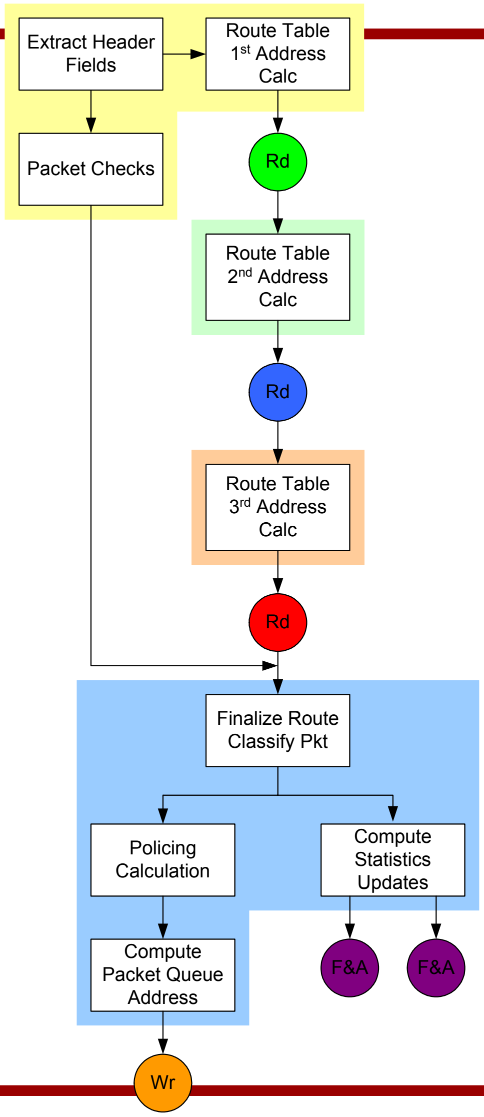
- Media applications (signal, image, video, packet, and graphics processing) are naturally expressed in this style
- Scientific applications can be efficiently cast as stream programs
- Others?
  - This is an open question
- Hypothesis
  - Any application with a long run time (large operation count) has a great deal of parallelism and hence can be cast as a stream program.

Example, Ingress Packet Functions

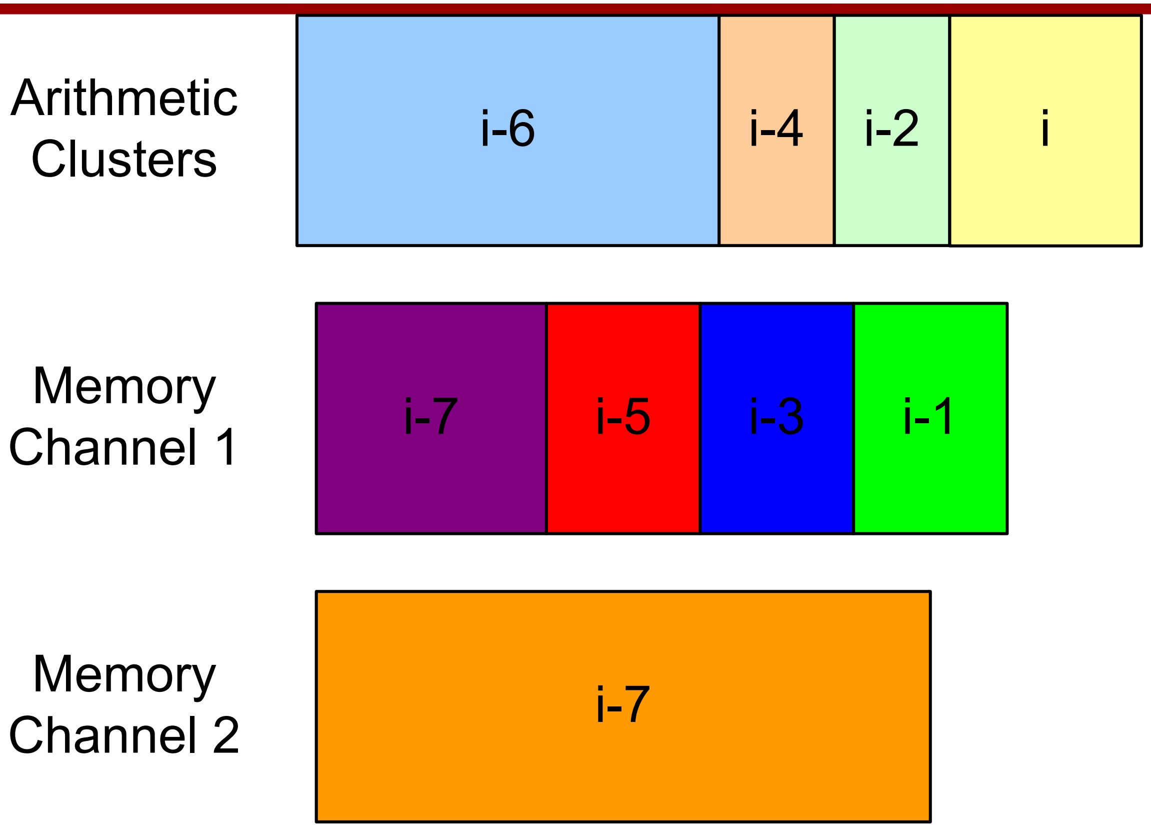


Extract Header Fields
Route Table 1st Address Calc
Packet Checks
Rd
Route Table 2nd Address Calc
Rd
Route Table 3rd Address Calc
Rd
Finalize Route Classify Pkt
Policing Calculation
Compute Statistics Updates
Compute Packet Queue Address
F&A
F&A
Wr

Split into Kernels


Extract Header Fields
Route Table 1st Address Calc
Packet Checks
Rd
Route Table 2nd Address Calc
Rd
Route Table 3rd Address Calc
Rd
Finalize Route Classify Pkt
Policing Calculation
Compute Statistics Updates
Compute Packet Queue Address
F&A
F&A
Wr

Software Pipeline The Loop



Each block represents a kernel or memory operation on a strip of packets

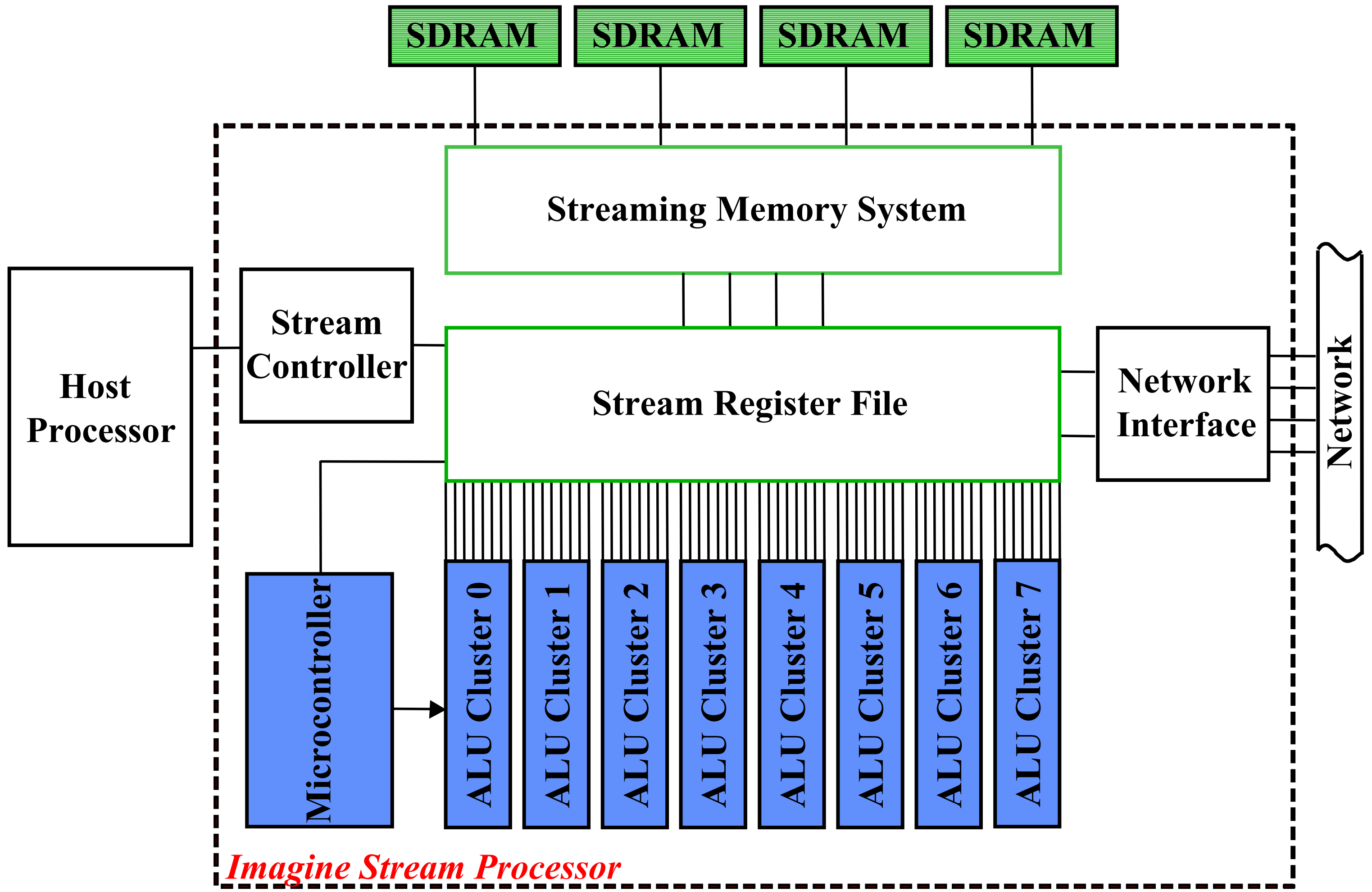
Is it hard to write stream programs?

- We will let you be the judge of that.
- It does constrain how you write a program
- Depends on the quality of the programming tools.

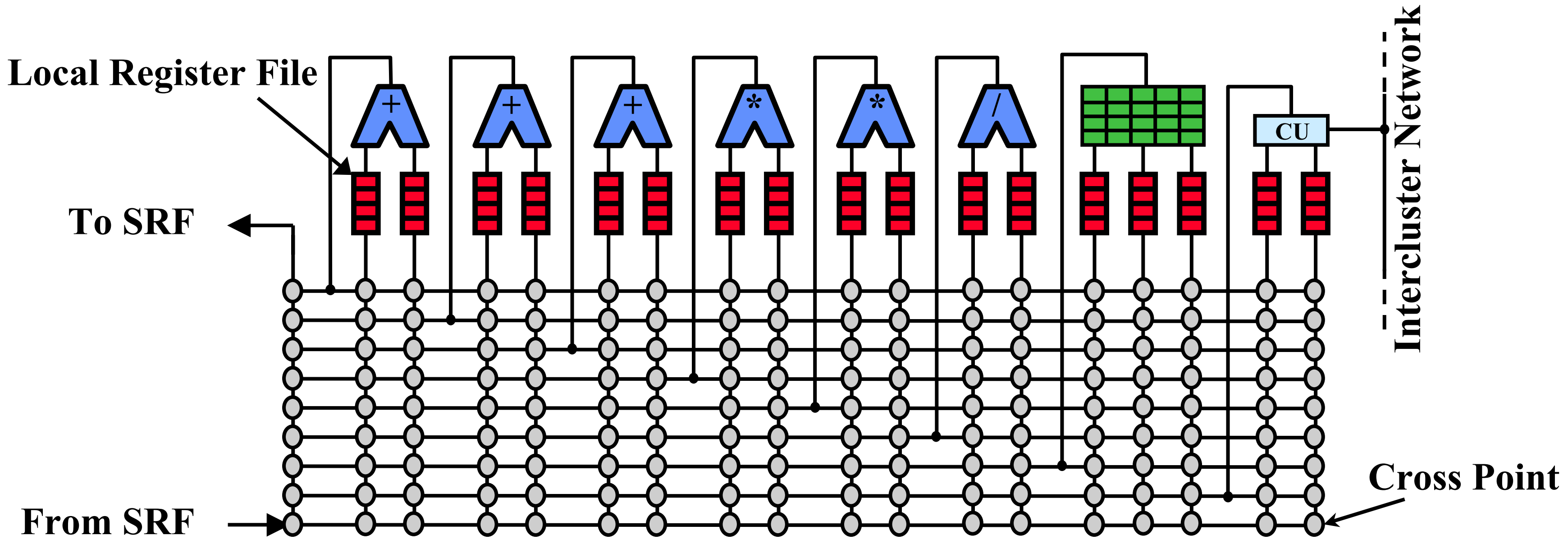
What is a Stream Processor?

- A processor that is optimized to execute a stream program
- Features include
  - Exploit parallelism
    - TLP with multiple processors
    - DLP with multiple clusters within each processor
    - ILP with multiple ALUs within each cluster
  - Exploit locality with a bandwidth hierarchy
    - Kernel locality within each cluster
    - Producer-consumer locality within each processor
- Many different possible architectures

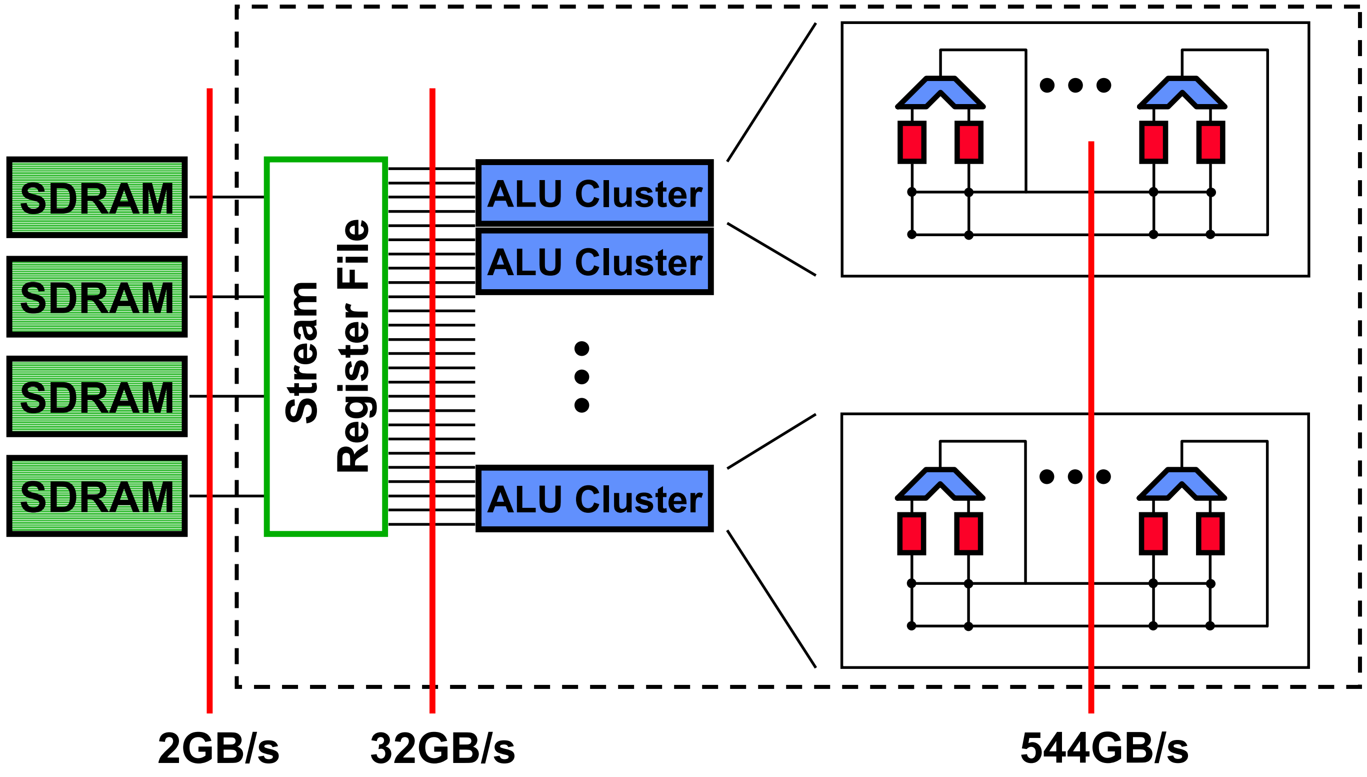
The Imagine Stream Processor



Arithmetic Clusters

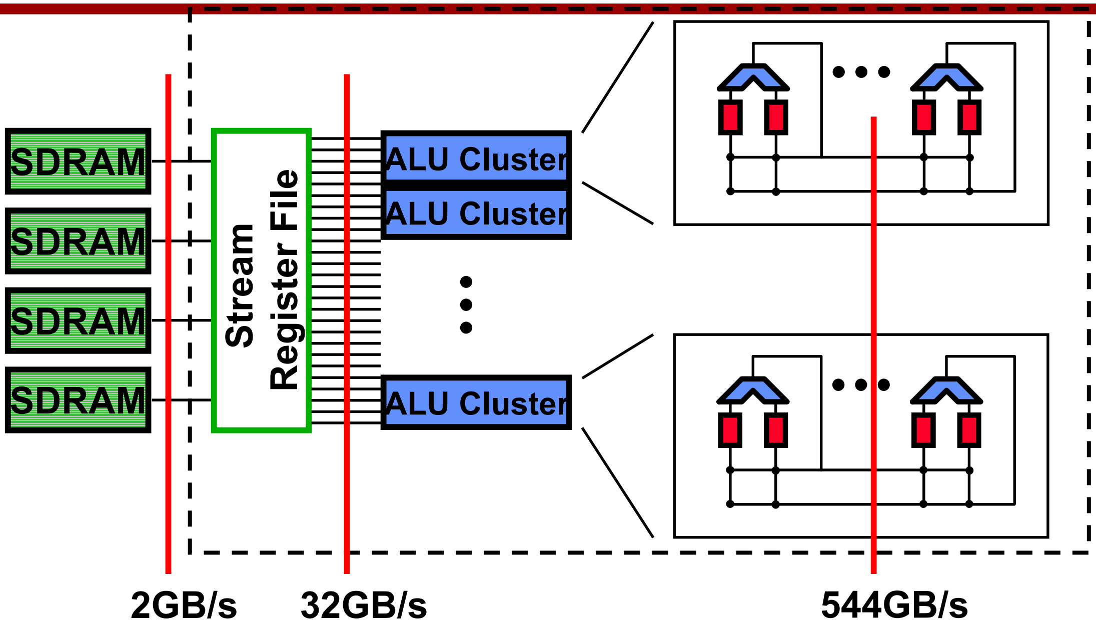


A Bandwidth Hierarchy exploits locality and concurrency



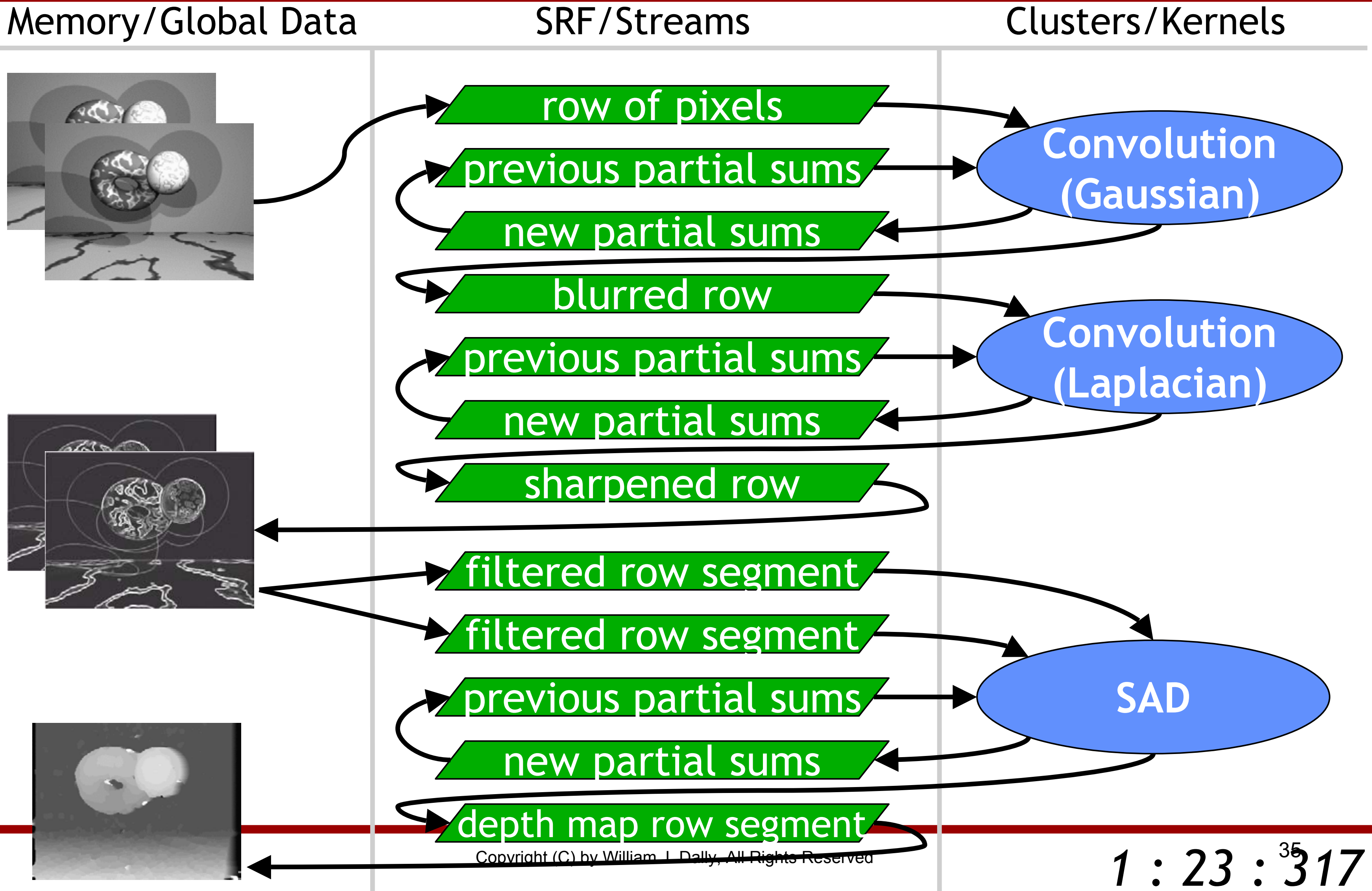
- VLIW clusters with shared control
- 41.2 32-bit floating-point operations per word of memory BW

A Bandwidth Hierarchy exploits kernel and producer-consumer locality



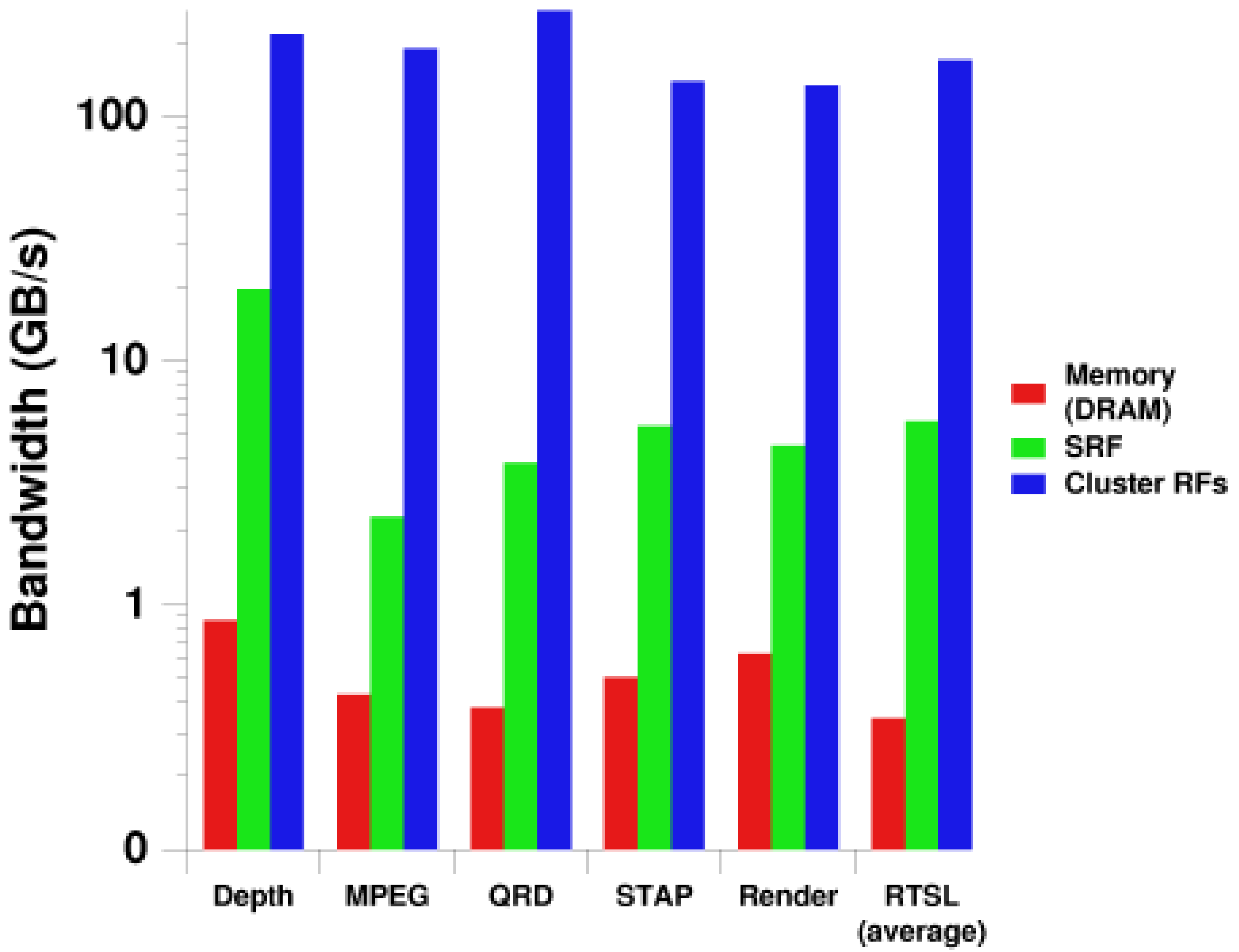
	Memory BW	***Global RF BW***	***Local RF BW***
Depth Extractor	0.80 GB/s	18.45 GB/s	210.85 GB/s
MPEG Encoder	0.47 GB/s	2.46 GB/s	121.05 GB/s
Polygon Rendering	0.78 GB/s	4.06 GB/s	102.46 GB/s
QR Decomposition	0.46 GB/s	3.67 GB/s	234.57 GB/s

Producer-Consumer Locality in the Depth Extractor

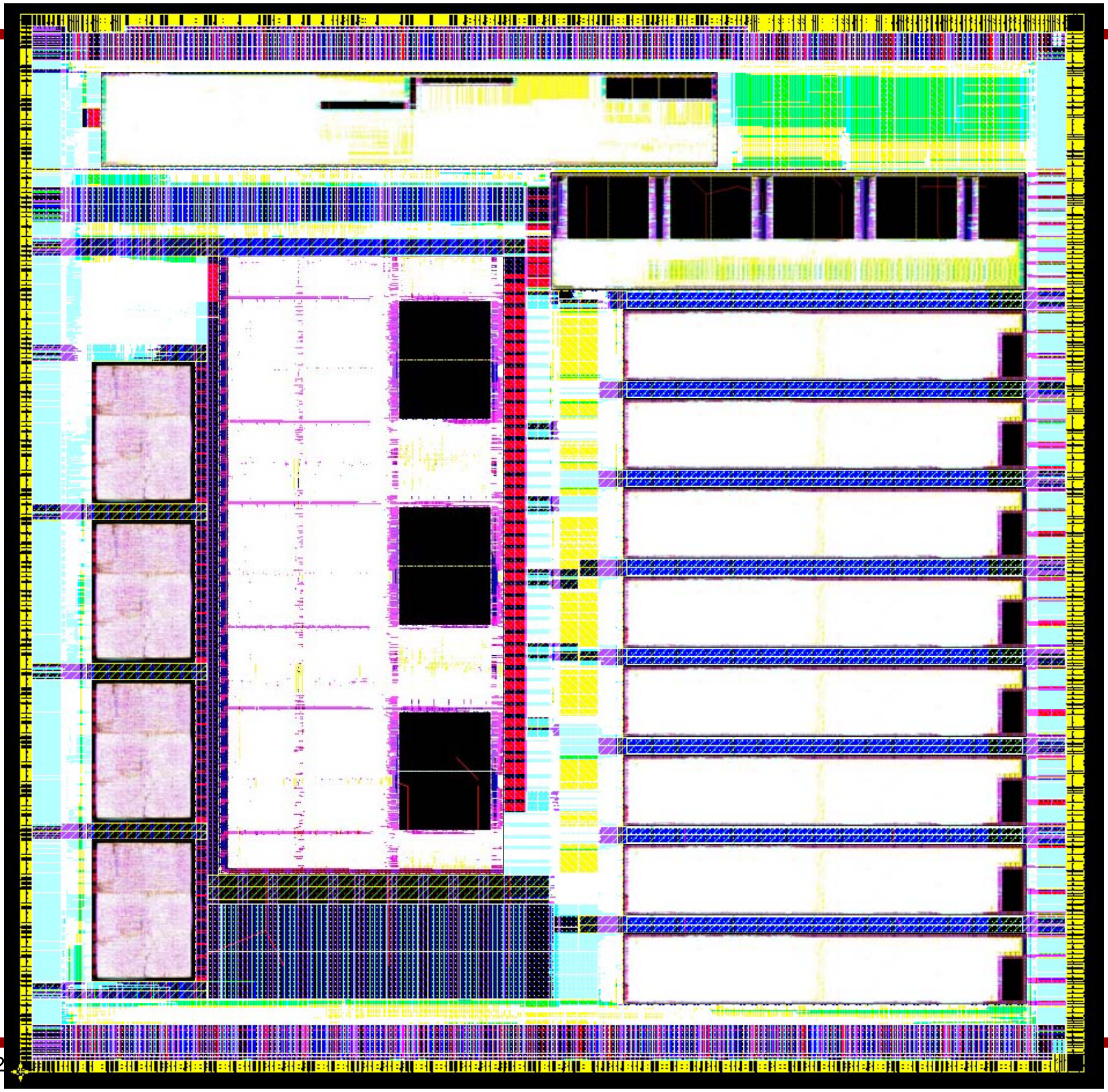


1 : 23 : 317

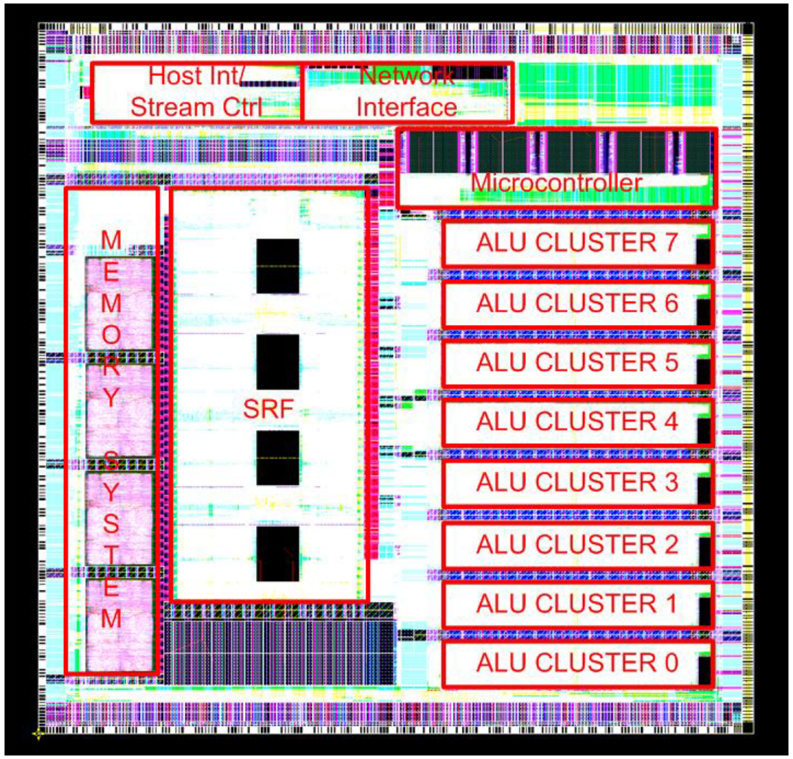
Bandwidth Demand of Applications

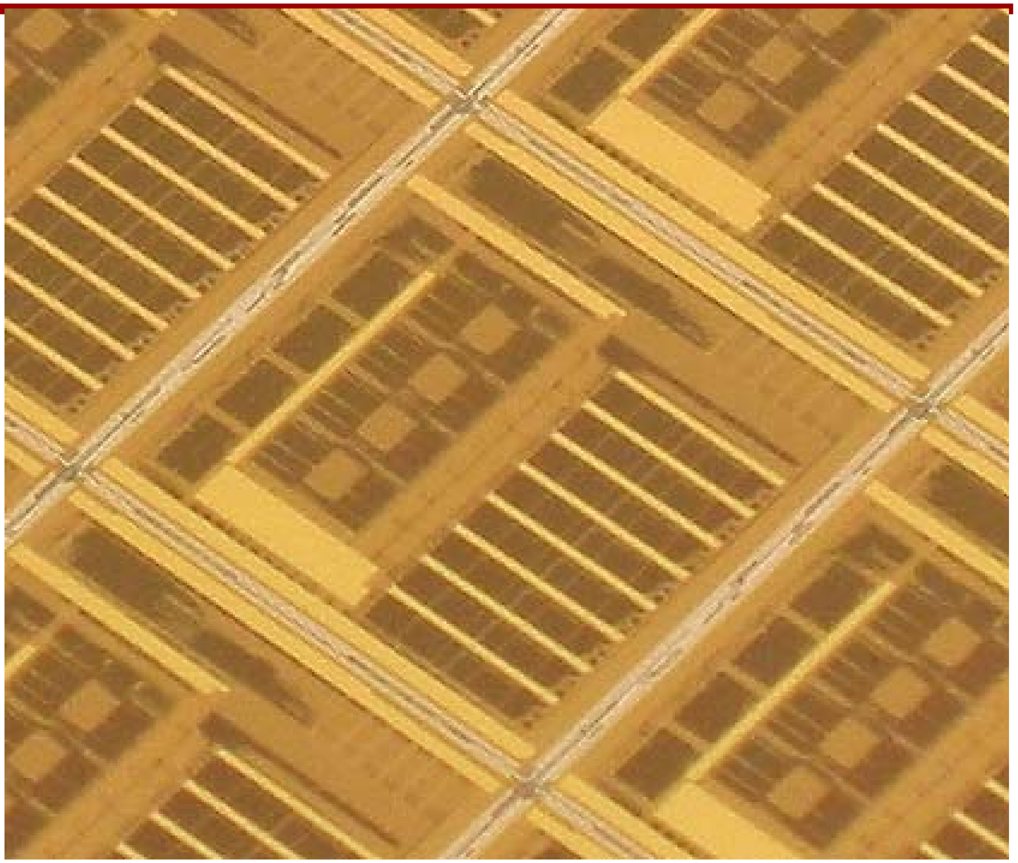


Overview



Die Photos

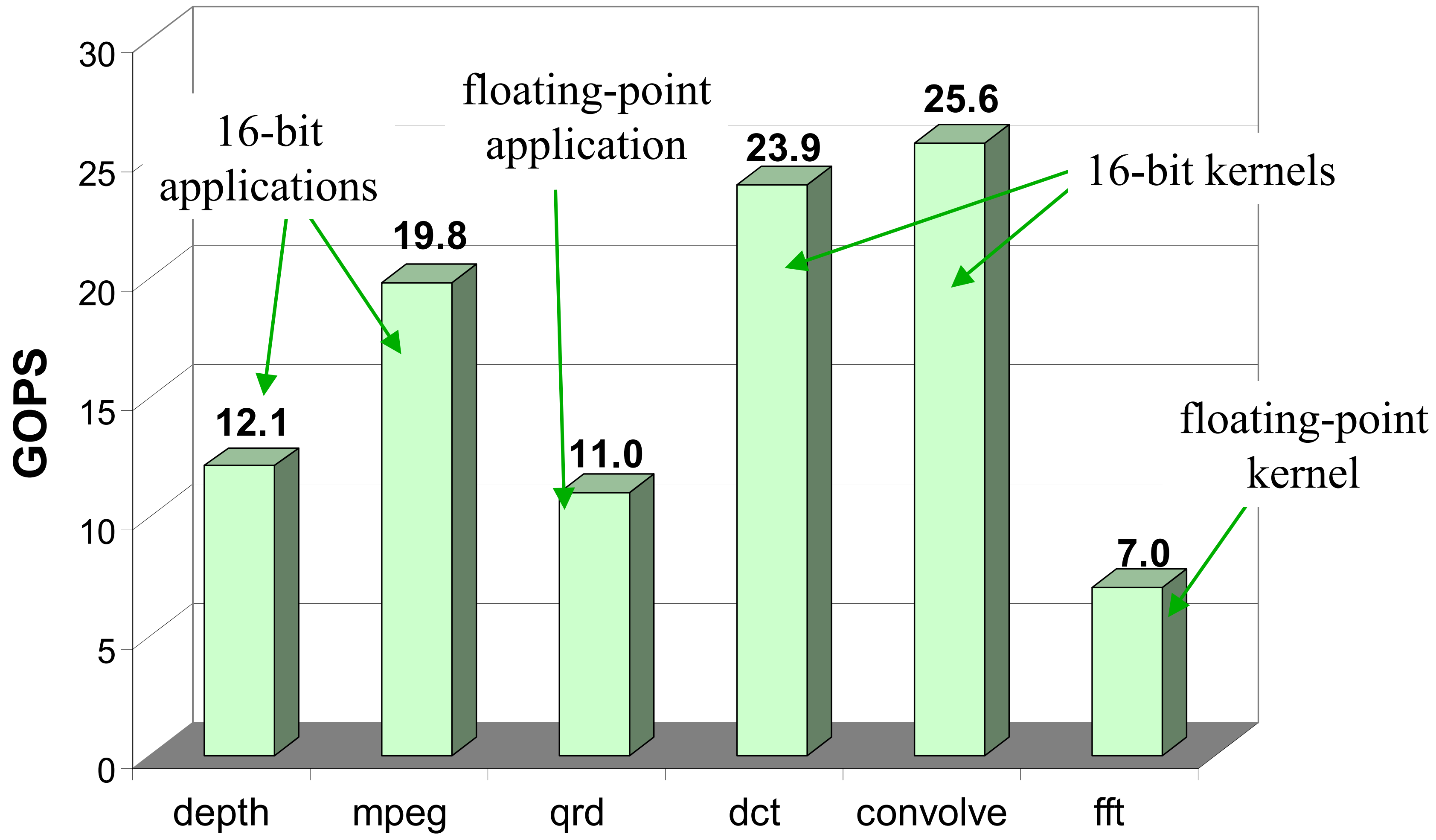




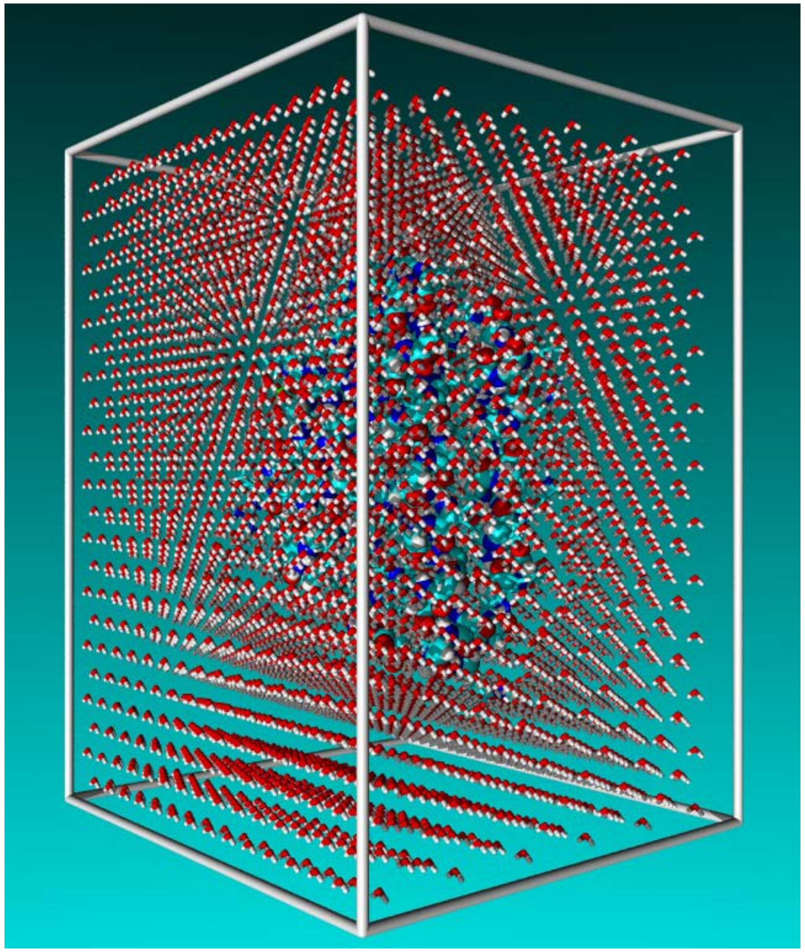
- 21 M transistors / TI 0.15μm 1.5V CMOS / 16mm x 16mm
- 300 MHz TTTT, hope for 400 MHz in lab
- Chips arrived 4/1/02, no fooling!

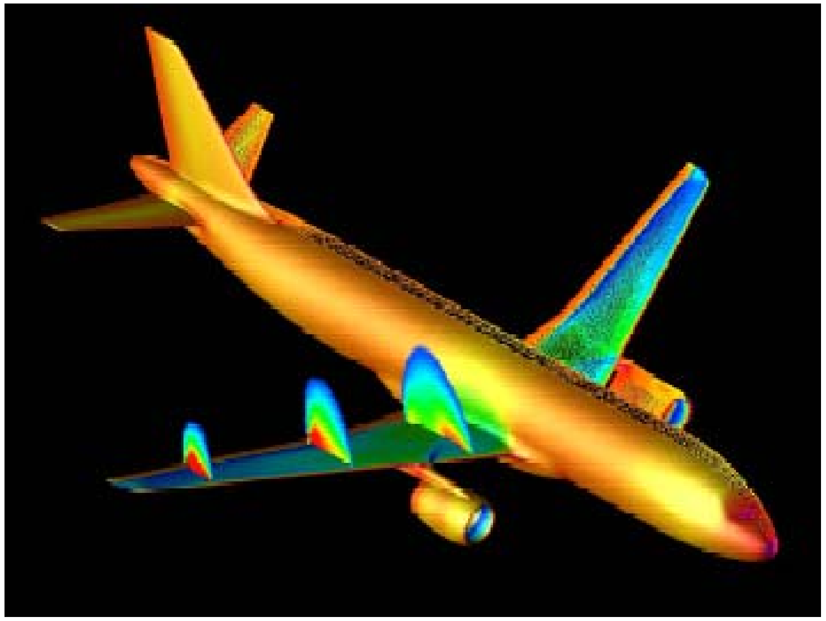
[Khailany et al., ICCD '02 (submitted)]

Performance demonstrated on signal and image processing

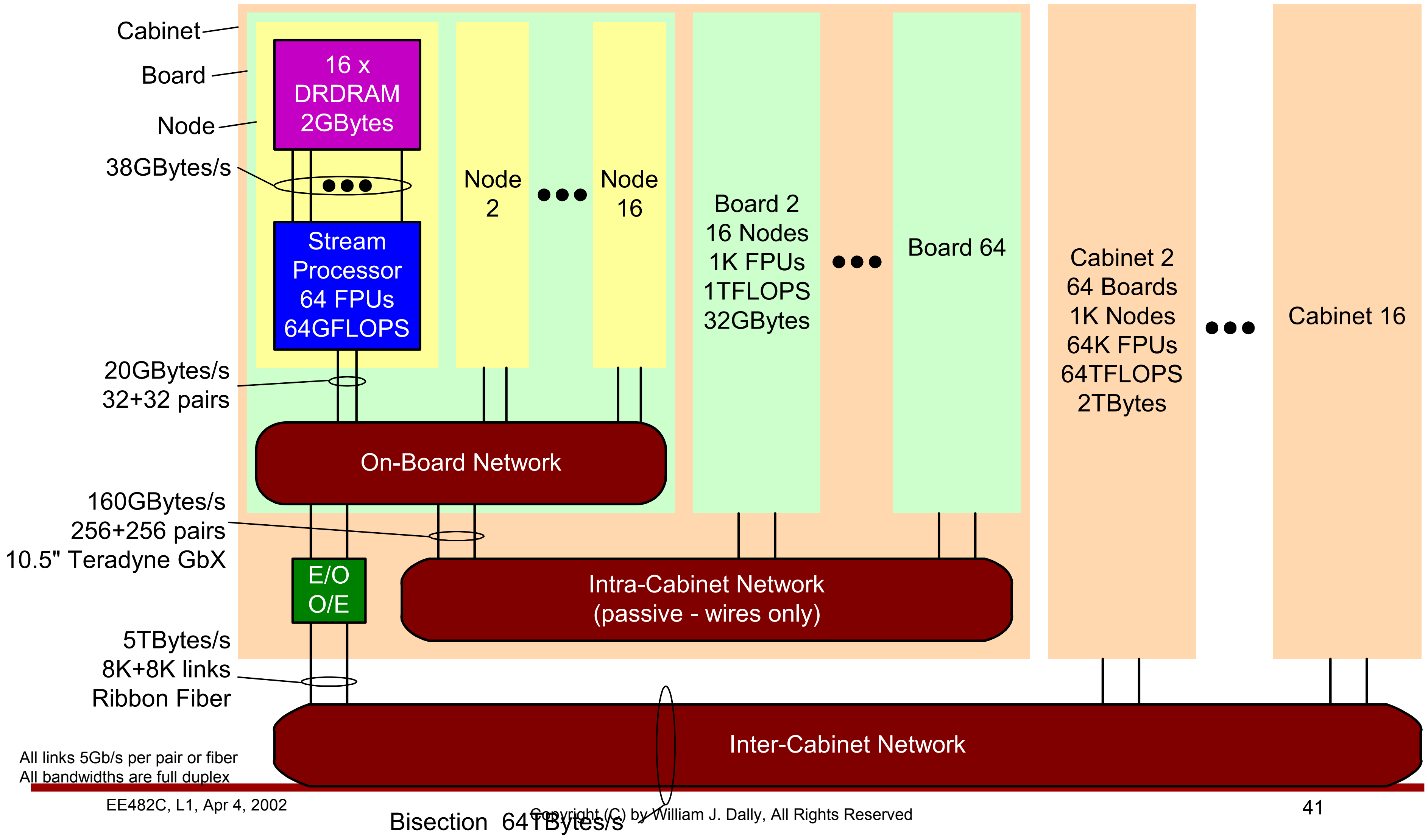


Initial studies indicate that it also applies to solving PDEs and ODEs





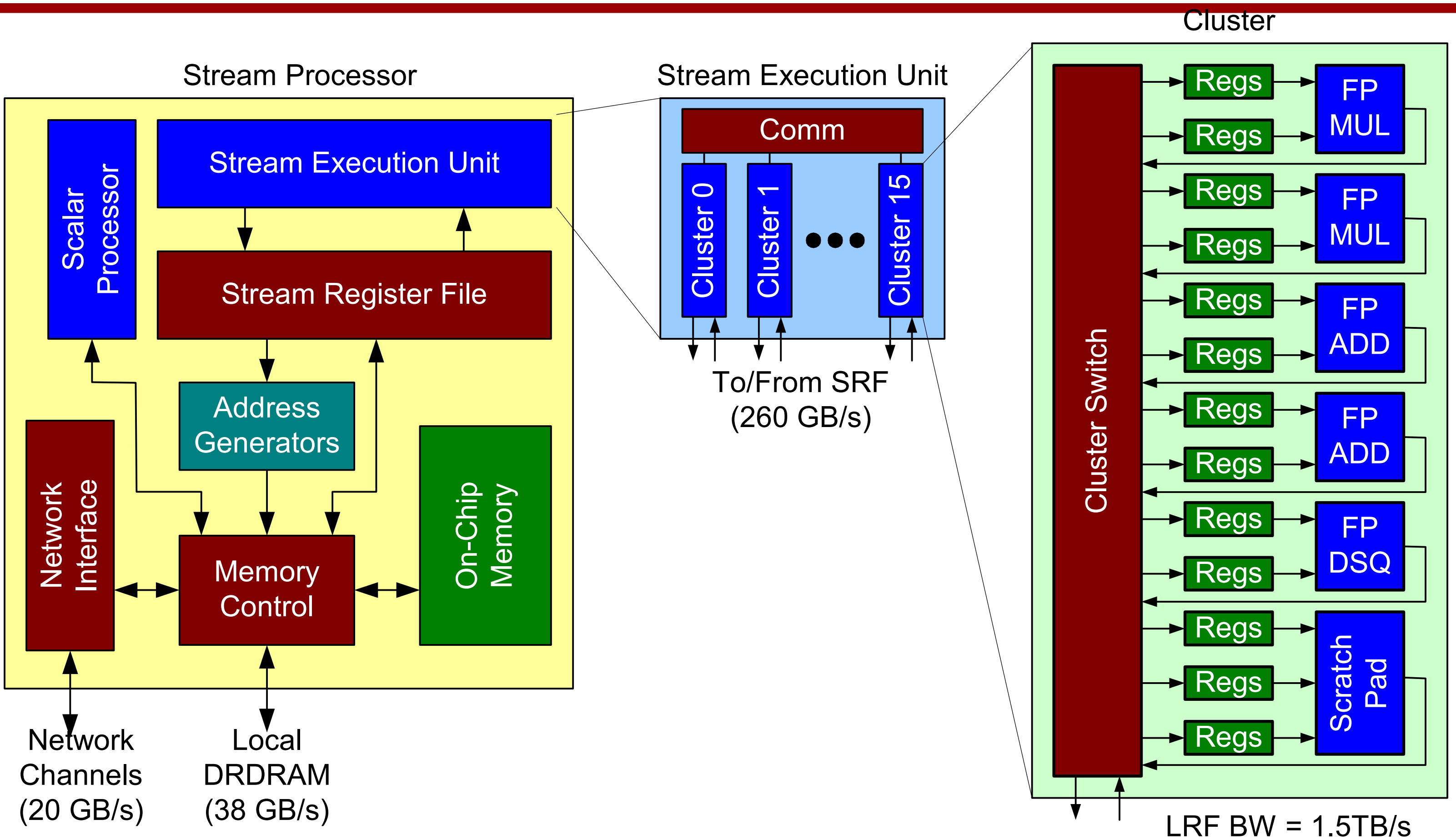
Architecture of a Streaming Supercomputer



Cabinet
Board
Node
16 x DRDRAM 2GBytes
38GBytes/s
Stream Processor 64 FPUs 64GFLOPS
Node 2
Node 16
20GBytes/s 32+32 pairs
On-Board Network
Board 2 16 Nodes 1K FPUs 1TFLOPS 32GBytes
Board 64
Cabinet 2 64 Boards 1K Nodes 64K FPUs 64TFLOPS 2TBytes
Cabinet 16
160GBytes/s 256+256 pairs 10.5" Teradyne GbX
E/O O/E
Intra-Cabinet Network (passive - wires only)
5TBytes/s 8K+8K links Ribbon Fiber
Inter-Cabinet Network
Bisection 64TBytes/s

All links 5Gb/s per pair or fiber
All bandwidths are full duplex

Streaming processor



Rough per-node budget

Item	Cost	Per Node
Processor chip	200	200
Router chip	200	50
Memory chip	20	320
Board/Backplane	3000	188
Cabinet	50000	49
Power	1	50
Per-Node Cost		976
$/GFLOPS (64/node)		15
$/M-GUPS (250/node)		4

Preliminary numbers, parts cost only, no I/O included.

Many open problems

- A small sampling
- Software
  - Program transformation
  - Program mapping
  - Bandwidth optimization
  - Conditionals
  - Irregular data structures
- Hardware
  - Alternative stream models
  - Register organization
  - Bandwidth hierarchies
  - Memory organization
  - Short stream issues
  - ISA design
  - Cluster organization
  - Processor organization

Next Time

- Discuss Imagine paper
- Discuss the stream programming model