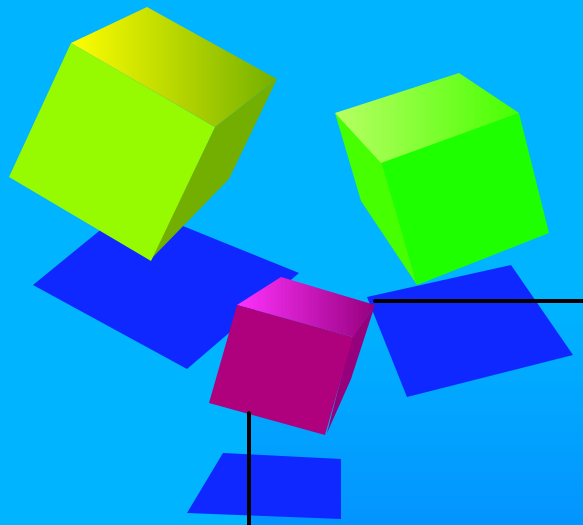
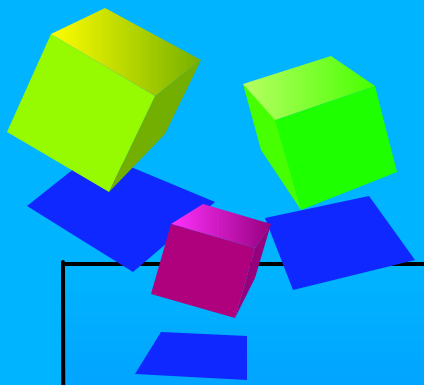


Stream Processing: The Imagine Architecture

Jason Dale

CS395T - Realtime Graphics

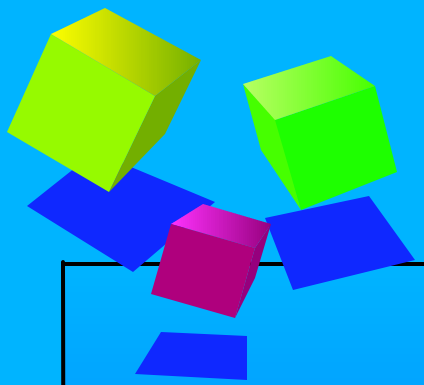
03/18/2003



Overview

- "Imagine: Media Processing With Streams", Khailany, et al.
  - Quick overview of Imagine
  - A word on streams
  - Imagine hardware
  - Streams on Imagine
  - Issues/Limitations of Streams
- "Efficient Conditional Operations for Data Parallel Architectures", Kapasi, et al.
  - Conditional Streams
  - Examples
- Discussion

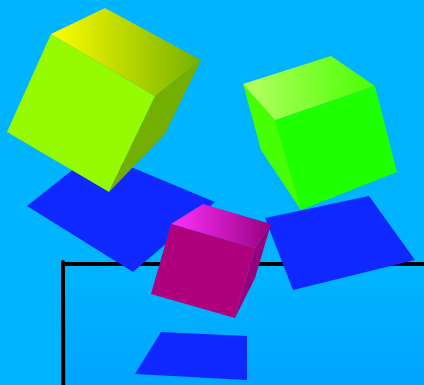
Several diagrams graciously provided by the Imagine team



Introduction to Imagine

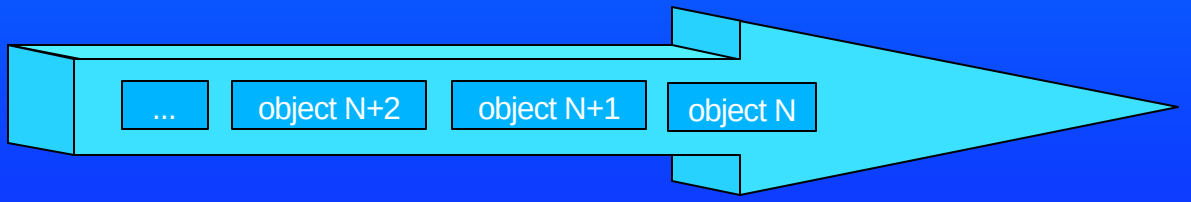
- Media / graphics domain
- Balance flexibility and performance
- Centered around stream processing
- Heritage: SIMD, Vector, IRAM, etc.

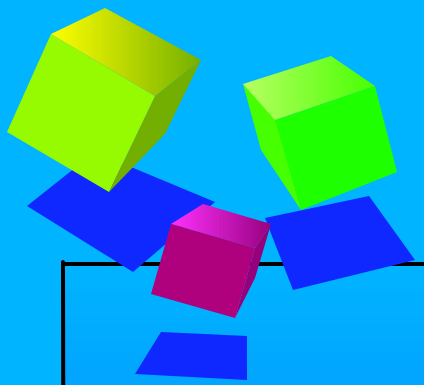
- What is Imagine?
  - Programming Model
  - Programming Languages
  - Architecture
  - Microarchitecture / implementation



A Word on Streams...

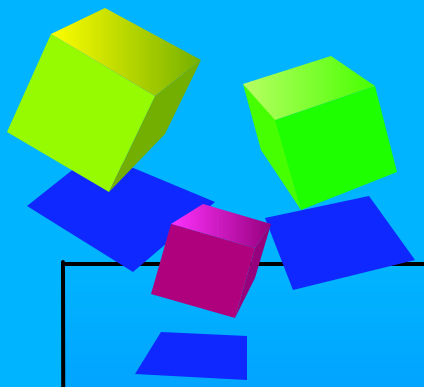
- Similiar to a queue, list, or array
  - Contains integers, polygons, "records", etc.
- Beginning and End?
- Serial or parallel?
- Ordering



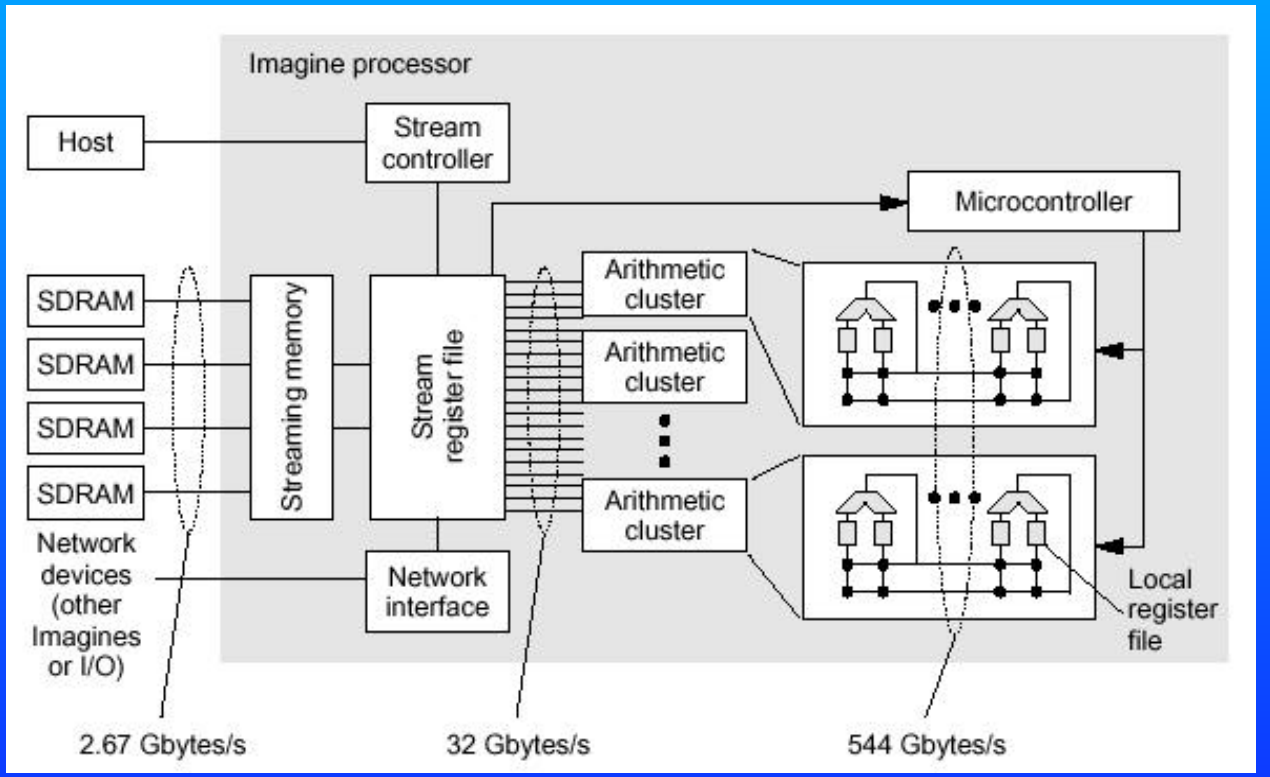


Imagine Hardware

- Major Hardware Components
  - VLIW Arithmetic Units (PEs)
  - Streaming Register File (SRF)
  - Stream Buffers (SBs)
  - Interfaces (memory, network, host)
  - Control (conditionals, programs, ucontroller...)
- Bandwidth Hierarchy
  - Website claims "architectural innovation"
  - The focus for scalability

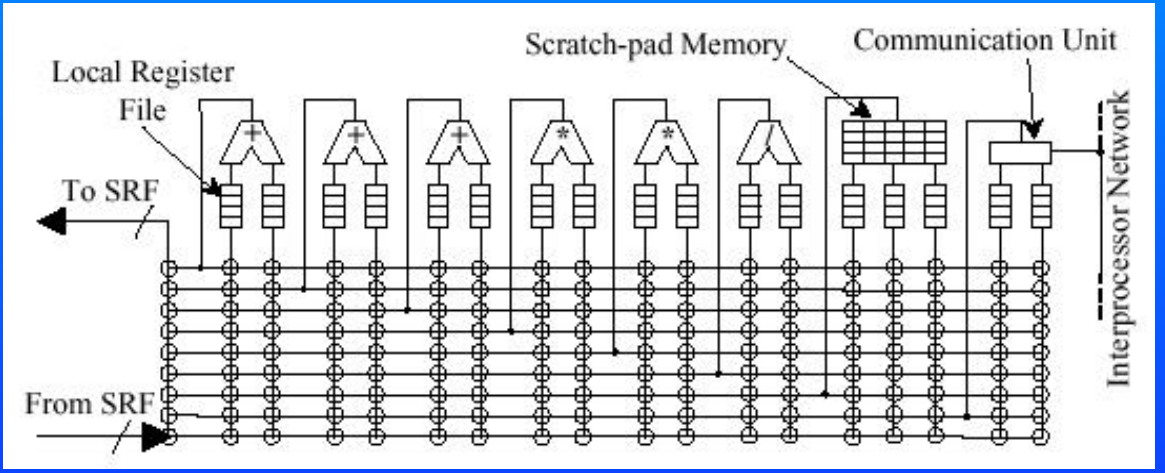


Hardware: Overview

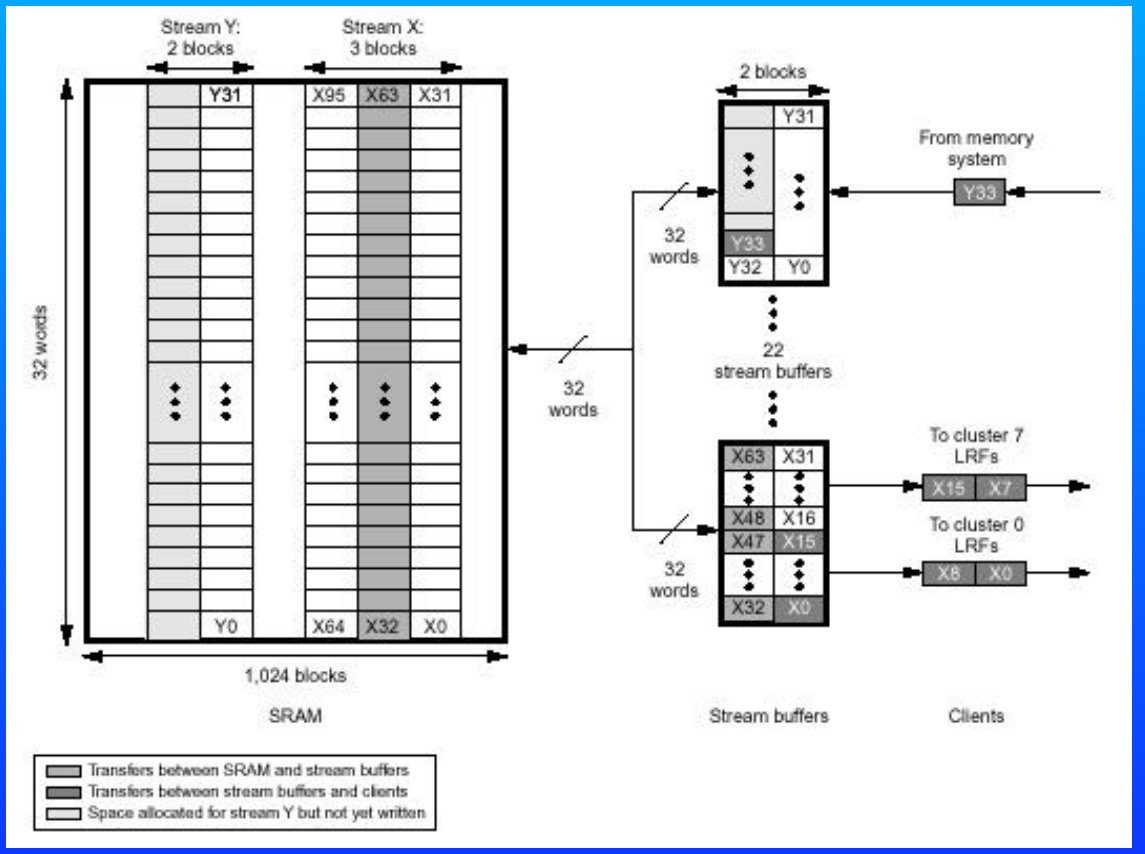


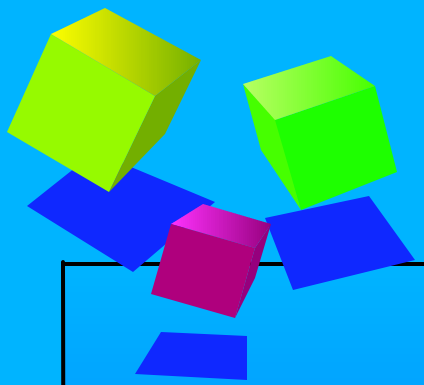
Hardware: PEs

- SIMD array of VLIW processors
- Decentralized registers - very nice!



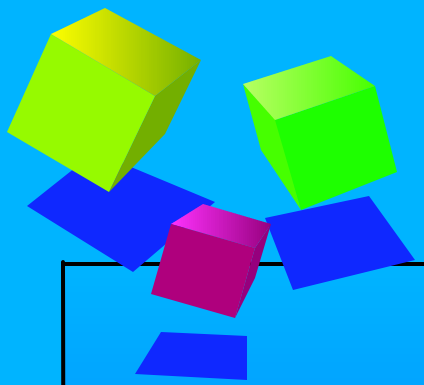
Hardware: SRF / SBs





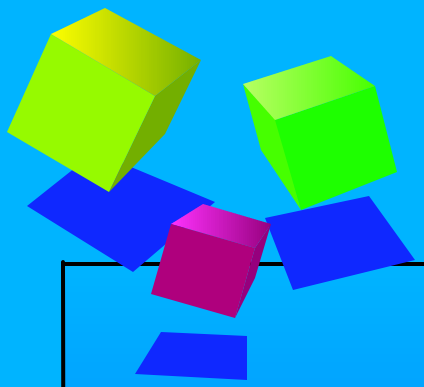
Bandwidth breakdown

- SRF: 25.6 GB/s
  - 2 cycles to fill/empty 1/2 SB @ 400MHz
  - Center of communication (cache/network)
  - Size not architecturally limited - great!
- SB: 30 GB/s
  - PEs: 8 * 1 words / cycle
  - Net: 8 * 2 words / cycle
  - Other: 6 * 1 words / cycle
- LRF: 435 GB/s (8 * 54.4 GB/s)
  - 15 Files * 2 ports
  - 272 words total / PE



Other Imagine Hardware

- Interfaces:
  - Host Processor
  - Memory - 2.7 GB/s (allows reordering)
  - Network - 4 GB/s (to other Imagine nodes)
- Microcontroller:
  - Distributes instructions to PEs
  - Performs control:
    - 1 bit feedback from each PE
    - While loops are the only control flow
    - No if/else - select and predicates only

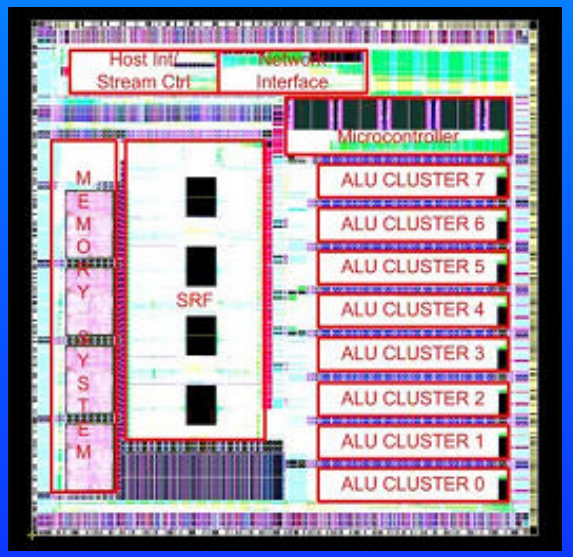


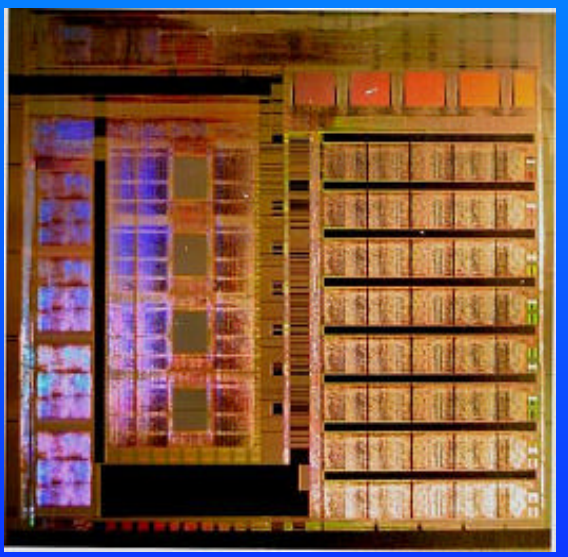
Imagine instructions

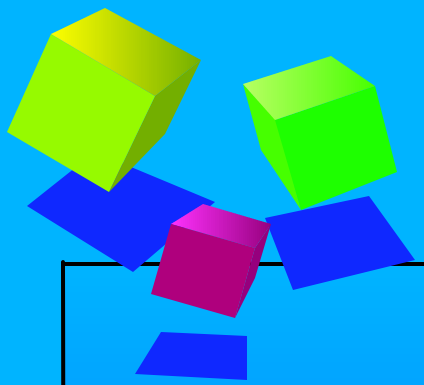
- Load Stream
- Store Stream
- Receive
- Send
- Cluster op (576 bit VLIW)
- Load microcode

Imagine Prototype - 2002

- TI 0.15u, 5 metals, Standard Cell
- 16mm^2, 21M Transistors, 1.5V
- 2.2 Watts (MPEG II - 18.3 GOPS)



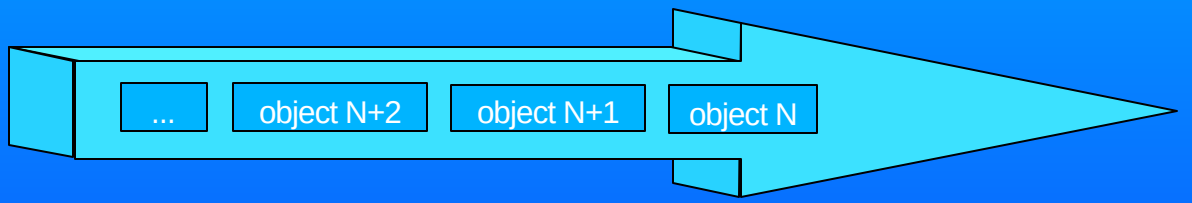




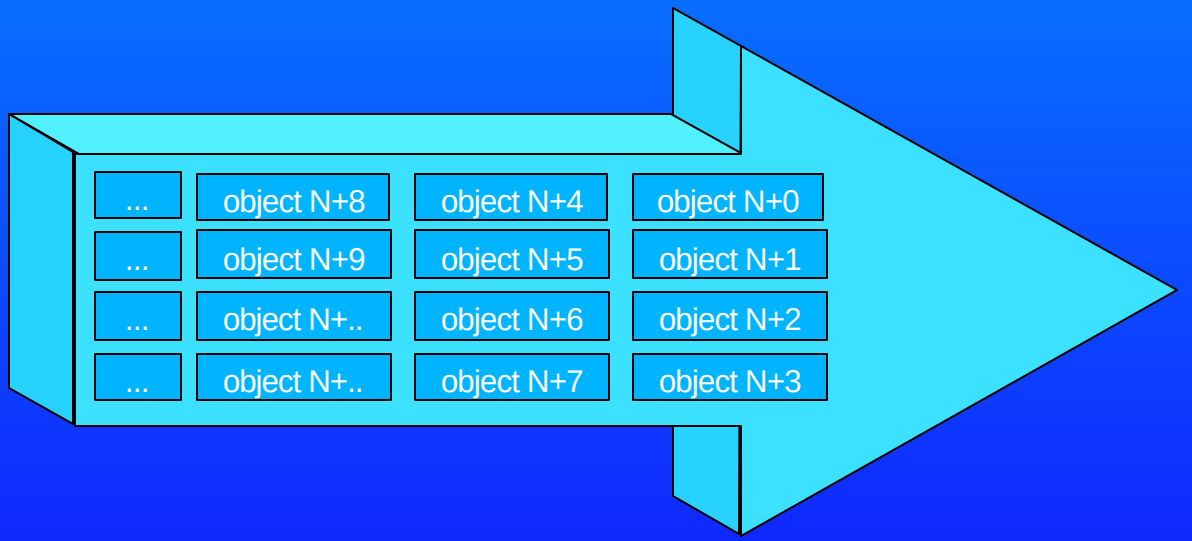
Streams on Imagine

- Serial and/or parallel
- 32k size limit

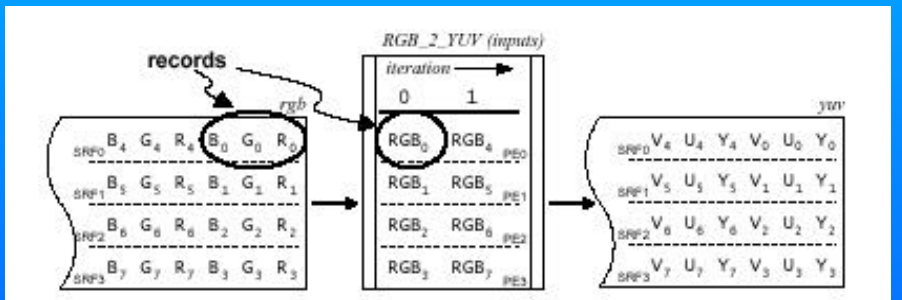
Logical:



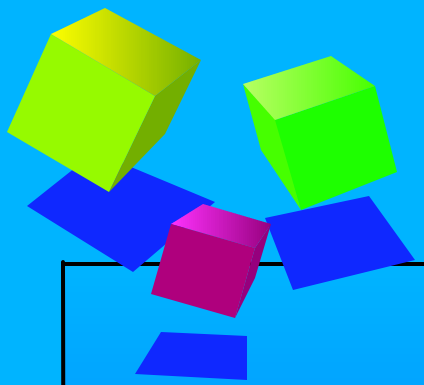
Physical:



Simple Example: RGB to YUV conversion

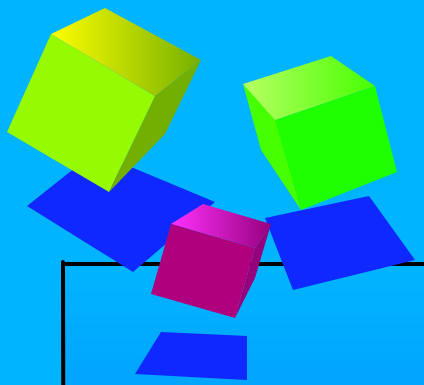



```
record RGB { char r, g, b; };
record YUV { char y, u, v; };

kernel RGB_2_YUV( istream<RGB> rgb,
                  ostream<YUV> yuv )
{
   // loops until all data in rgb is read
   loop_until ( rgb.empty() ) {
      rgb >> in;      // Get next el. from rgb stream
      out.y = C1*in.r + C2*in.g + C3*in.b;
      out.u = C4*(in.b - out.y);
      out.v = C5*(in.r - out.y);
      yuv << out;     // Append out onto the yuv stream
   }
}
```



Conditional Streams

- RGB to YUV
  - No control flow
  - No inter-record communication
  - No data-dependant timing
  - 1 in / 1 out - same ordering (Vertex Shader)
- Issues:
  - Streams may be logically serial (ordered)
  - Processed as SIMD
    - Slowest PE determines speed (Control Flow)
    - Null outputs
    - Multiple outputs

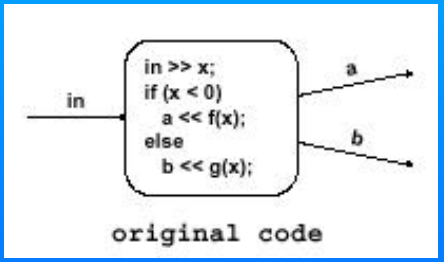


Conditional Streams

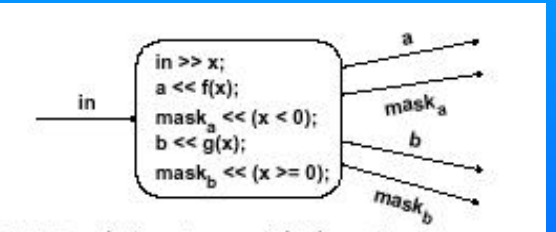
- Paper # 2 - Conditional Streams

- The gist of conditional streams
- Conditional Switching
  - For compressing streams
  - Filter example
- Conditional Combining
  - For expanding streams
  - Interleave example
- Load Balancing
  - For variable time (data-dependant control)
  - Decrementer example

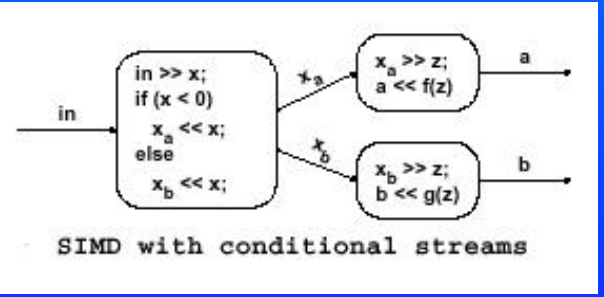
Conditional Streams



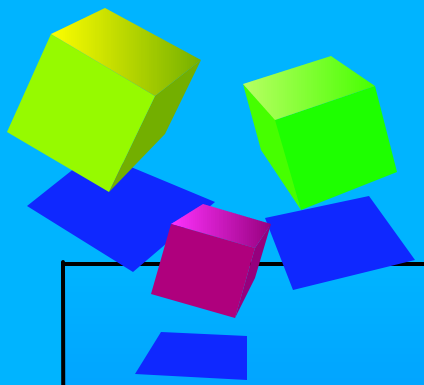
original code



SIMD without conditional streams



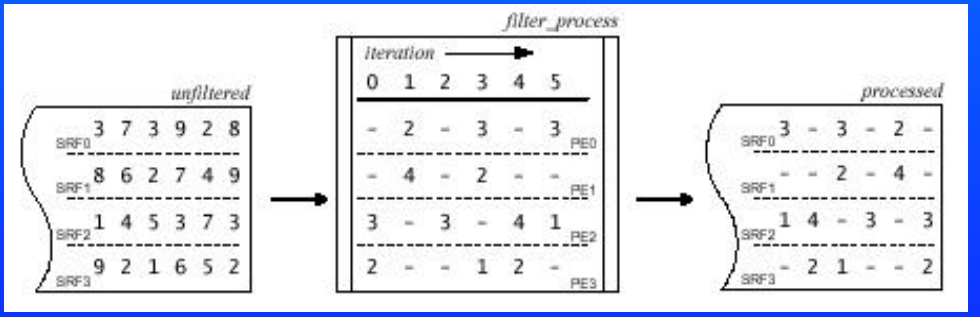
SIMD with conditional streams

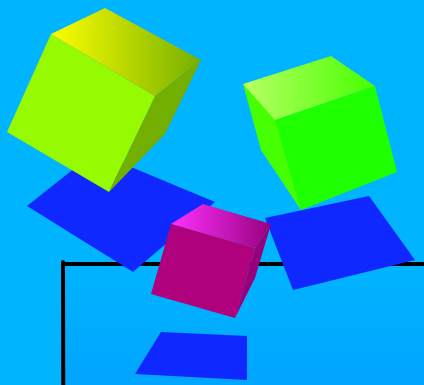


Conditional Switching: Filtering Example

- Without conditonal streams:

```
kernel filter_process( istream<int> unfiltered,
                       ostream<bool> mask,
                       ostream<int> processed )
{
   loop_until ( unfiltered.empty() ) {
      unfiltered >> curr;
      valid = (curr <= 4);
      mask << valid;
      processed << compute(curr);
   }
}
```



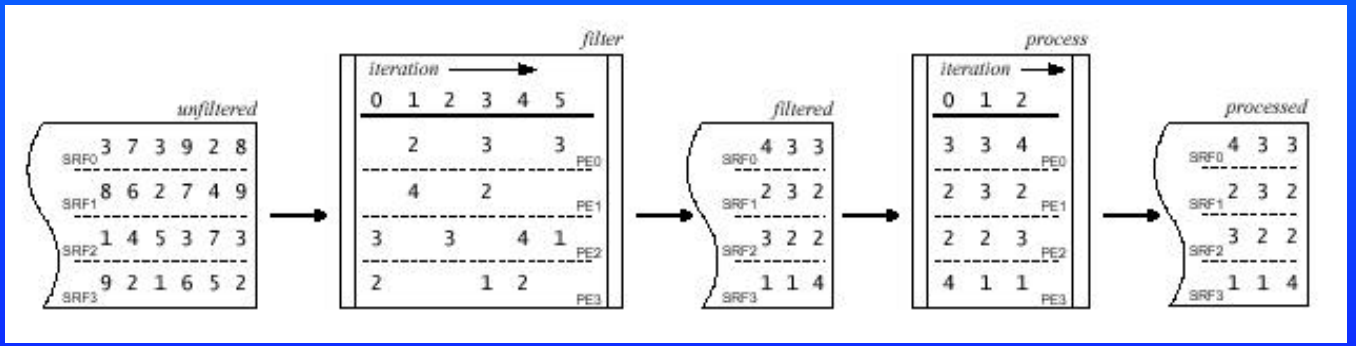


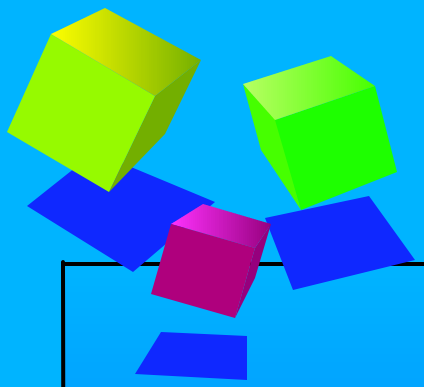
Conditional Switching: Filtering Example

- With conditional streams:

```
kernel filter( istream<int> unfiltered,
               costream<int> filtered  )
{
   loop_until ( unfiltered.empty() ) {
      unfiltered >> curr;
      valid = (curr <= 4);
      filtered(valid) << curr;
   }
}
```

```
kernel process( istream<int> filtered,
                ostream<int> processed )
{
   loop_until ( filtered.empty() ) {
      filtered >> curr;
      processed << compute(curr);
   }
}
```





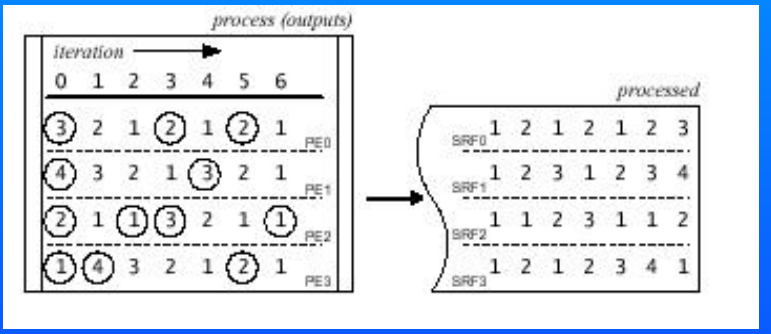
Conditional Combining: Interleave Example

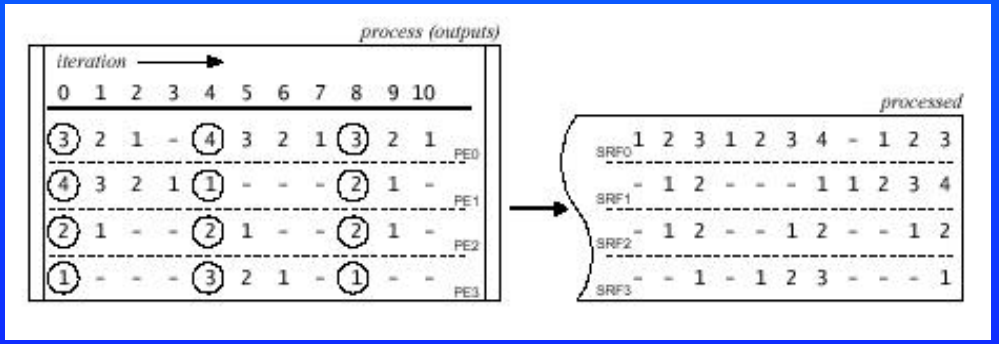
```
kernel interleave( istream<bool> case,
                   int addrA, int addrB,
                   ostream<unsigned int> loadIdx )
{
   loop_until ( case.empty() ) {
      case >> sel;

      // ACnt = # of PEs below you in which sel==1
      // BCnt =    "      "    "    "     sel==0
      // Note: PE_i is 'below' PE_j if (i < j)
      ACnt = numBelow(sel);  BCnt = MY_ID - ACnt;
      myAddr = sel ? (ACnt + addrA) : (BCnt + addrB);

      // numA calc. by broadcasting highest PE's val
      numA = broadcast(NUM_PE-1, ACnt + (sel ? 1 : 0) );
      addrA += numA; addrB += NUM_PE - numA;

      loadIdx << myAddr;
   }
}
```

```
kernel interleave(istream<bool> case,
                  cistream<int> inA,
                  cistream<int> inB,
                  ostream<int> out )
{
   // assume case.len == inA.len + inB.len
   loop_until ( case.empty() ) {
      case >> sel;
      inA(sel) >> a;
      inB(!sel) >> b;
      out << (sel ? a : b);
   }
}
```

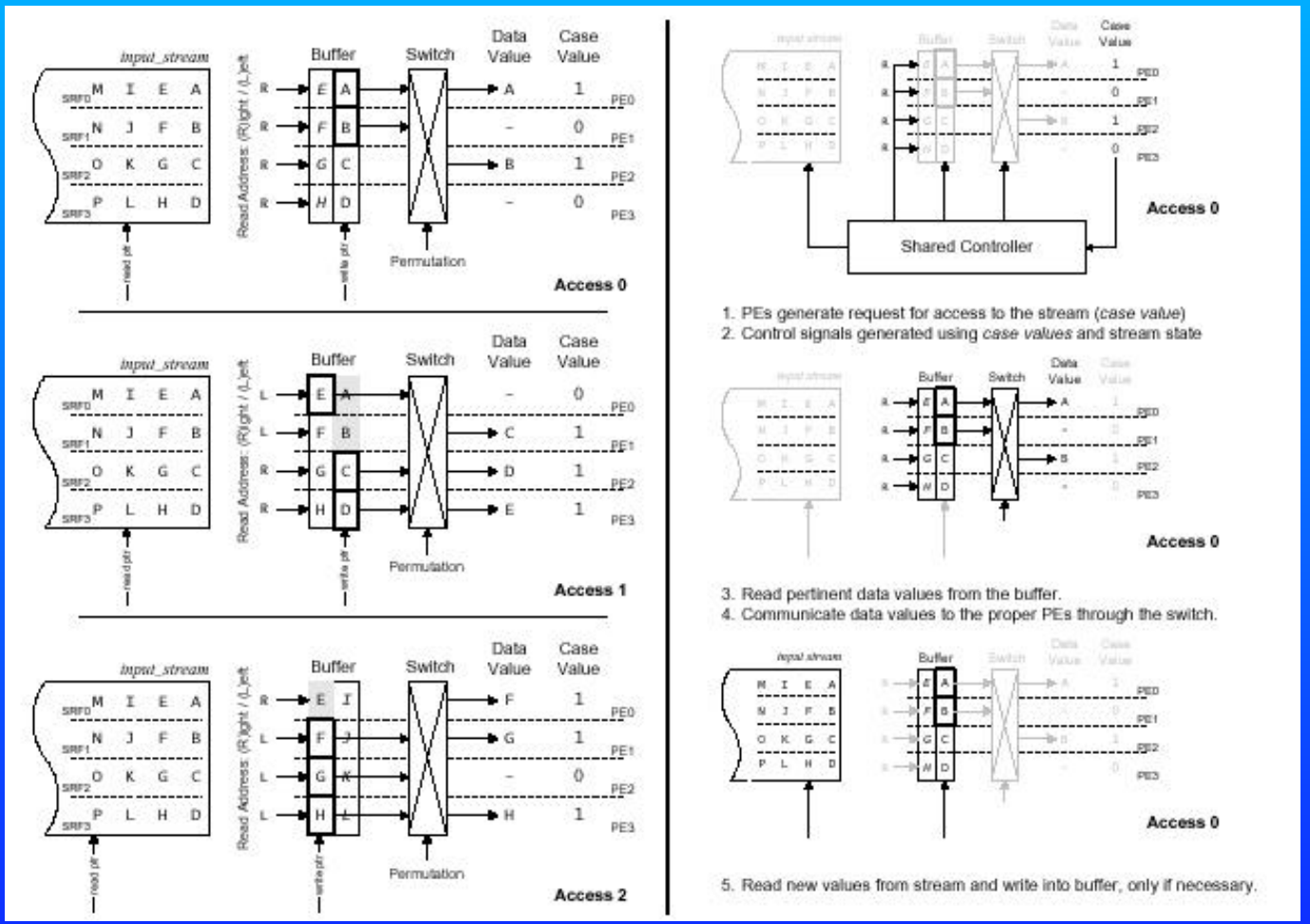
Load Balancing

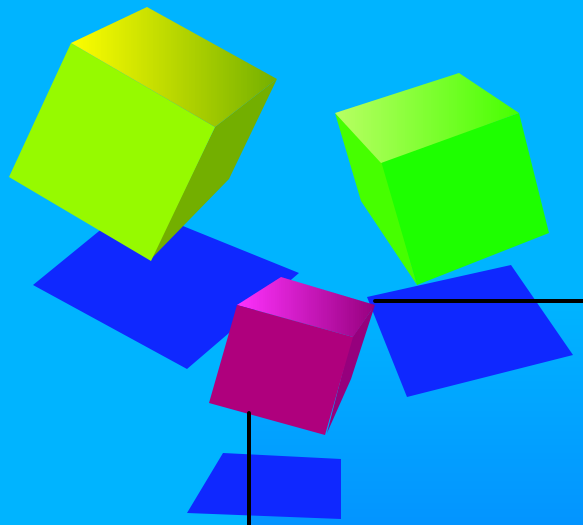
- Interesting Example...
  - Demonstrates the mechanism
  - Would it work this well in practice?
  - Seems a bit contrived
  - Any thoughts/opinions?





Hardware:

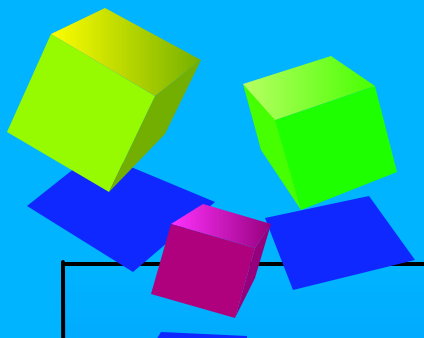

input_stream
Buffer
Switch
Data Value
Case Value
Read Address: (R)ight / (L)eft
Permutation
Access 0
Access 1
Access 2
Shared Controller
1. PEs generate request for access to the stream (case value)
2. Control signals generated using case values and stream state
3. Read pertinent data values from the buffer.
4. Communicate data values to the proper PEs through the switch.
5. Read new values from stream and write into buffer, only if necessary.



Discussion

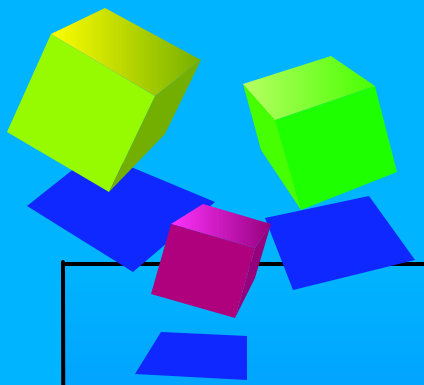
Imagine: Discussion Questions

- Programming Environment
  - StreamC - Application level
  - KernelC - Lower level
  - Dynamic runtime environment
- Mapping programs to stream model
  - ☺ Signal/image processing, Graphics, etc.
    - Fixed dataflow, regular structure, small record sizes
  - 😐 Quicksort, Big FFTs
    - Regular structure? record size?
  - ☹ Big Random Data, Streams - Network Processing?
    - Memory behavior, dynamic/variable structure
- Streams seem unlimited at first
  - Size restrictions? (Streams, Objects, etc...)
  - Polygon rendering (why not vertices?)
  - How are records partitioned?



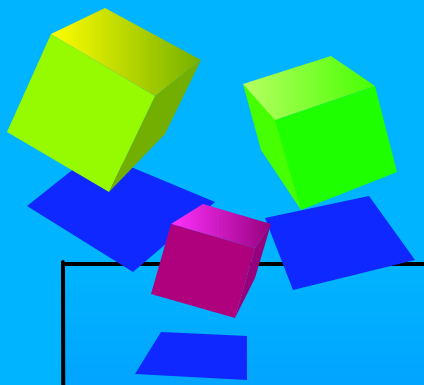
Imagine: Discussion Questions

- Dynamic streams?
  - Creation and destruction
  - Dynamic # of readers
  - Sorting question, again
  - Stream Management (Caching / Spilling / Balancing)
- Scalability
  - Repartitioning kernels
  - A million PEs? A million Imagine chips? Coherency?
  - Flexibility vs. overhead
- Architecture flexibility
  - VLIW - upgrade path? Intermediate representation?
  - Stream size / Double buffering?
  - Realtime issues?
    - Dynamic stream scheduling
    - Host



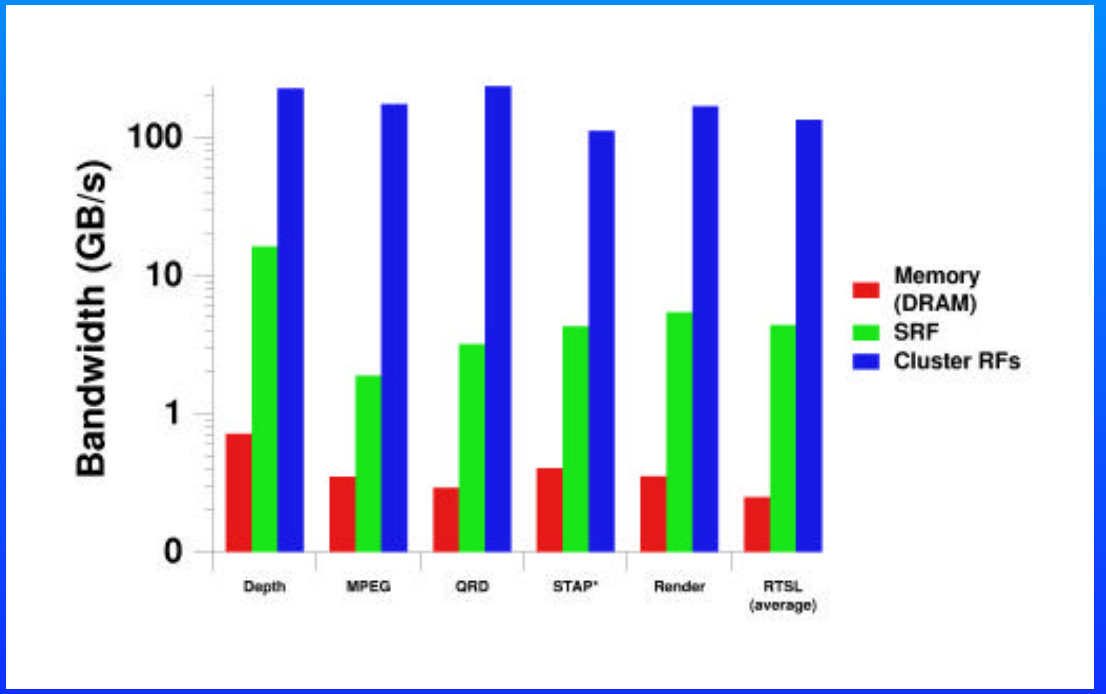
Conditional Streams: Discussion

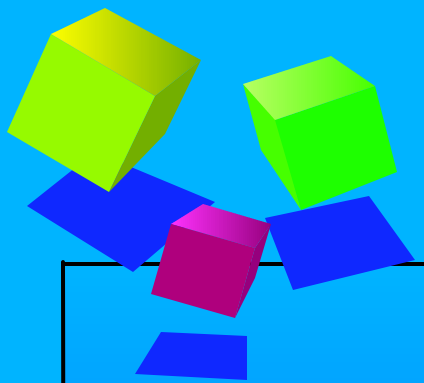
- Large record sizes?
- Compared to D.F. systems at different granularities:
  - Bit level (ASIC/FPGA)
  - Word level (Traditional DF machines)
  - Stream level
- Compared to traditional multiprocessors:
  - Message Passing
  - Shared Memory
  - What about for streaming workloads?



Discussion: Imagine Performance

- From the Imagine Website:
  - Remember the old "MIPS" metric?





Discussion: Imagine Performance

- Another view (from website):

	Arithmetic Bandwidth	Application Performance
Applications		
Stereo Depth Extraction	11.92 GOPS (16-bit)	320x240 8-bit gray scale at 198 fps
MPEG-2 Encoding	15.35 GOPS (16- and 8-bit)	320x288 24-bit color at 287 fps
QR Decomposition	10.46 GFLOPS	192x96 matrix decomposition in 1.44 ms
Polygon Rendering	5.91 GOPS (floating-point and integer)	35.6 fps for 720x720 "ADVS" benchmark
Polygon Rendering with Real-Time Shading Language	4.64 GOPS (floating-point and integer)	16.3M pixels/second; 11.1M vertices/second
Kernels		
Discrete Cosine Transform	22.6 GOPS (16-bit)	34.8 ns per 8x8 block (16-bit)
7x7 Convolution	25.6 GOPS (16-bit)	1.5 us per row of 320 16-bit pixels
FFT	6.9 GFLOPS	7.4 us per 1,024-point floating-point complex FFT

  - MPEG-2 encoding: 720x480 @ 105 FPS