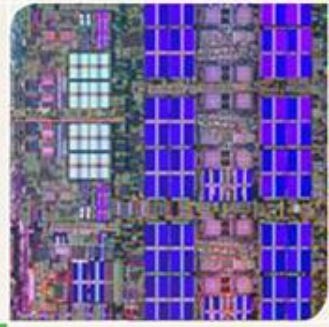
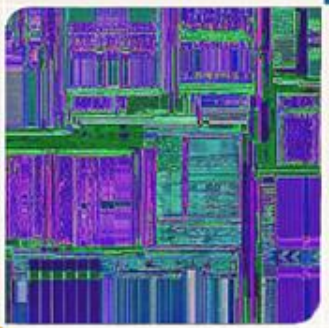
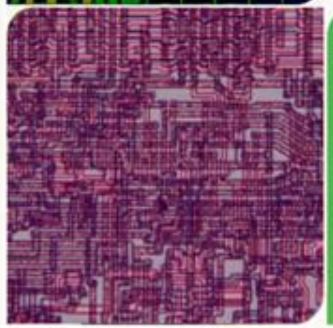



CS4803DGC Design and Programming of Game Console

Spring 2011

Prof. Hyesoon Kim

Georgia Tech College of Computing



Lectures are from

- Reading assignment [LRB]
- http://www.ece.neu.edu/groups/nucar/GPGPU/GPGPU-2/Seiler.pdf



Motivation for LRB

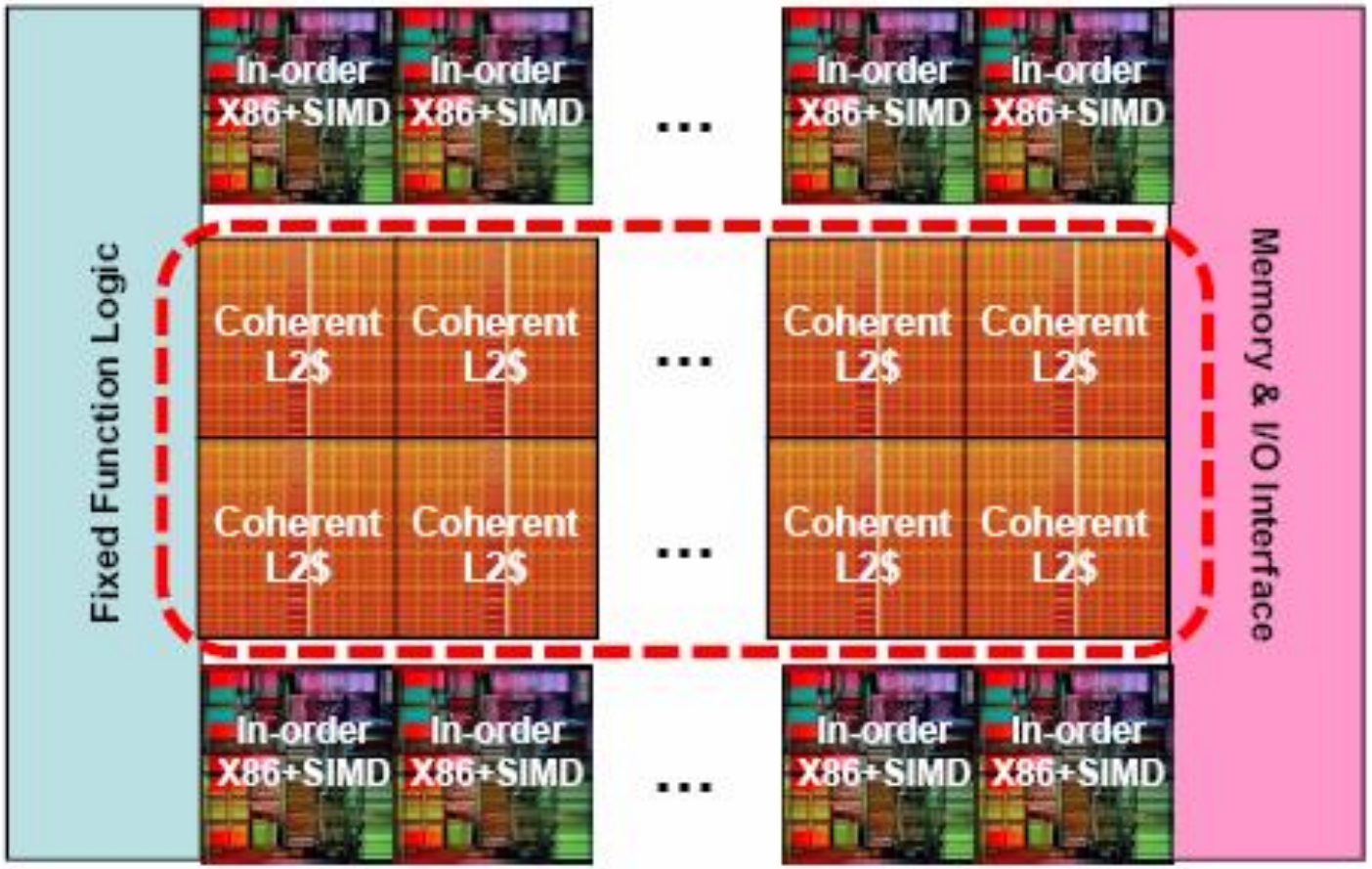
- Economic reason: Game market is getting bigger
- Technology reason: Power wall, memory wall. → Heterogeneous architecture shows promising results
- Programmable Graphics



In-order vs. OOO

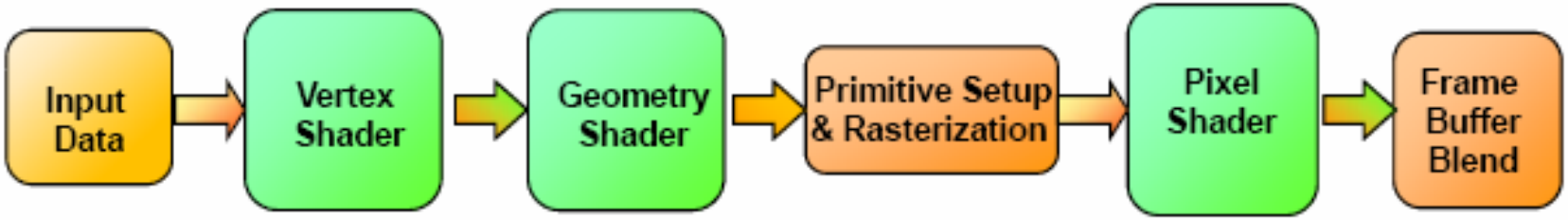
# CPU cores:	2 out-of-order	10 in-order
Instruction issue:	4 per clock	2 per clock
VPU per core:	4-wide SSE	16-wide
L2 cache size:	4 MB	4 MB
Single-stream:	**4 per clock**	**2 per clock**
Vector throughput:	**8 per clock**	**160 per clock**

Larrabee Architecture

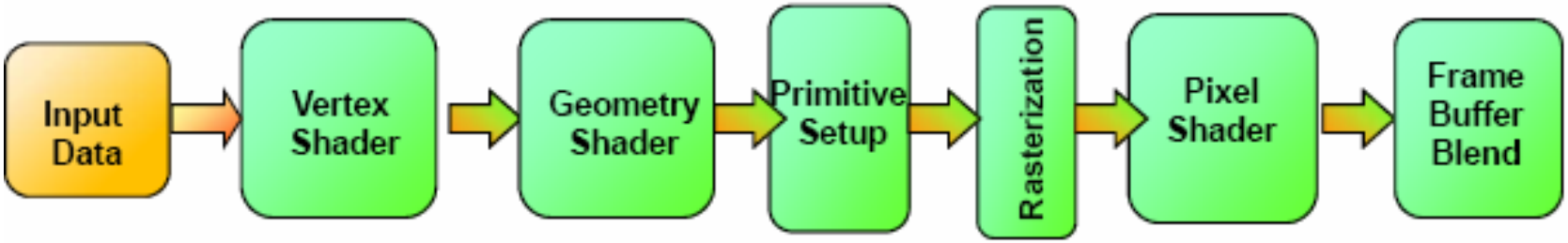


- 4-way SMT in order processor with cache coherence
- Extended X86 ISA
- Fixed functions: texture filtering

Programmable Pipeline Comparison



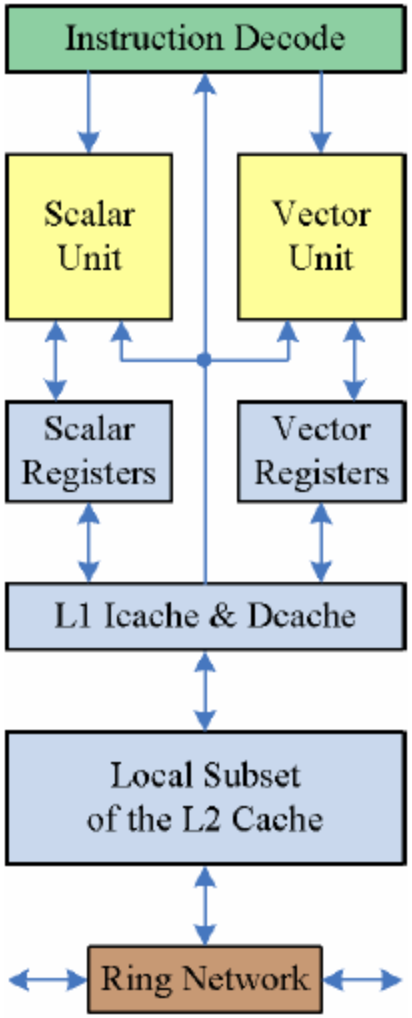
Conventional GPGPU pipeline (base on DirectX10)



Larrabee's fully programmable pipeline

http://users.ece.gatech.edu/~lanterma/mpg08/Larrabee_ECE4893.pdf

Core



- Pentium processor in-order
- Extended X86 (64bit, new instructions)
- 4-way SMT
- 32KB I-cache, 32KB D-cache (statically partitioned)
- 256KB Locale L2 cache (subset of L2 ache)

Dual issue

- U-pipe V-pipe
- Primary pipeline: All instructions (U-pipe)
- Secondary: Limited instructions (V-pipe)

  loads, stores, simple ALU operations, branches, cache manipulation instructions, and vector stores.
- Reply on compiler's paring
  - VLIWish again



Cache managements

- Use cache as extended register file storage
- Target for Stream applications
- Each core can

  – Fast-access its local subset of L2 (256KB) – Access other's L2 shares too

  • Control for non-temporal streaming data (SSE)

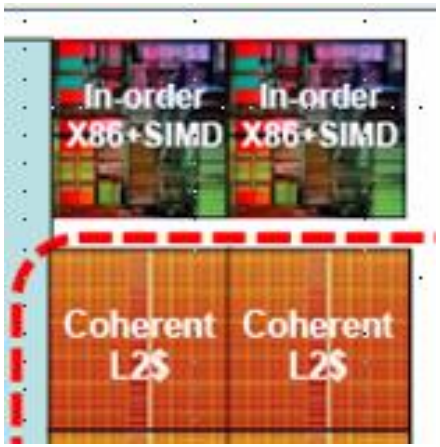
  • Prefetch to L1, or L2 only

  •Mark a streaming cache line for early eviction

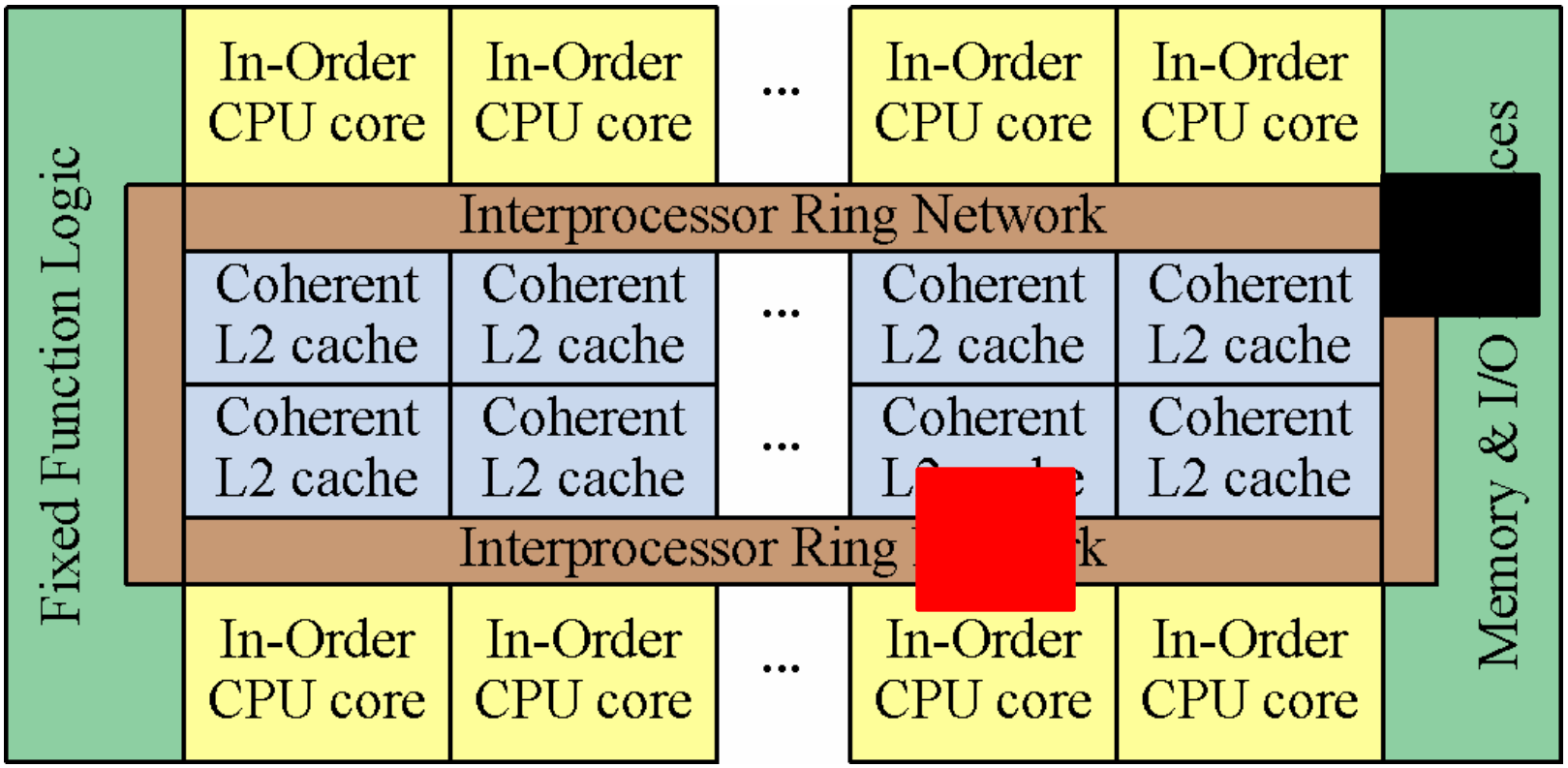
  •Render target kept in L2

L2-Cache and Ring network

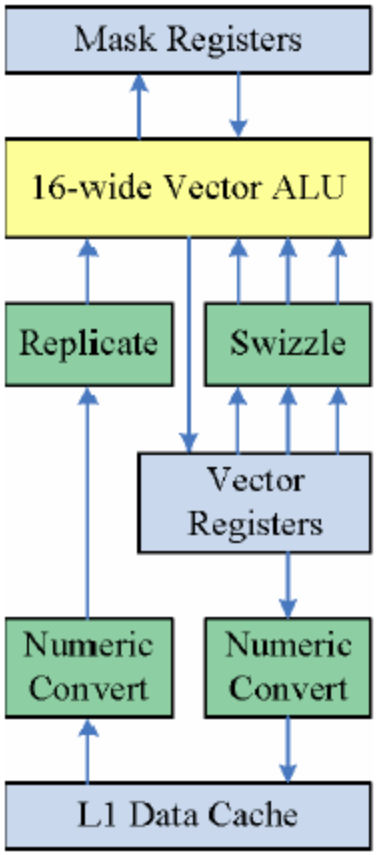
- Global L2 cache is divided into 256KB Local L2 per core
- Data written by a CPU core is stored in its own L2 cache subset and is flushed from other subsets, if necessary
- Bi-direction Ring network (<16)
  - Even cycle, odd cycle: one clock per one hop
- Each ring data-path is 512-bits wide per direction
- L2 cache Insertion requires cache coherence checking
- Memory and fixed function access



Ring Network



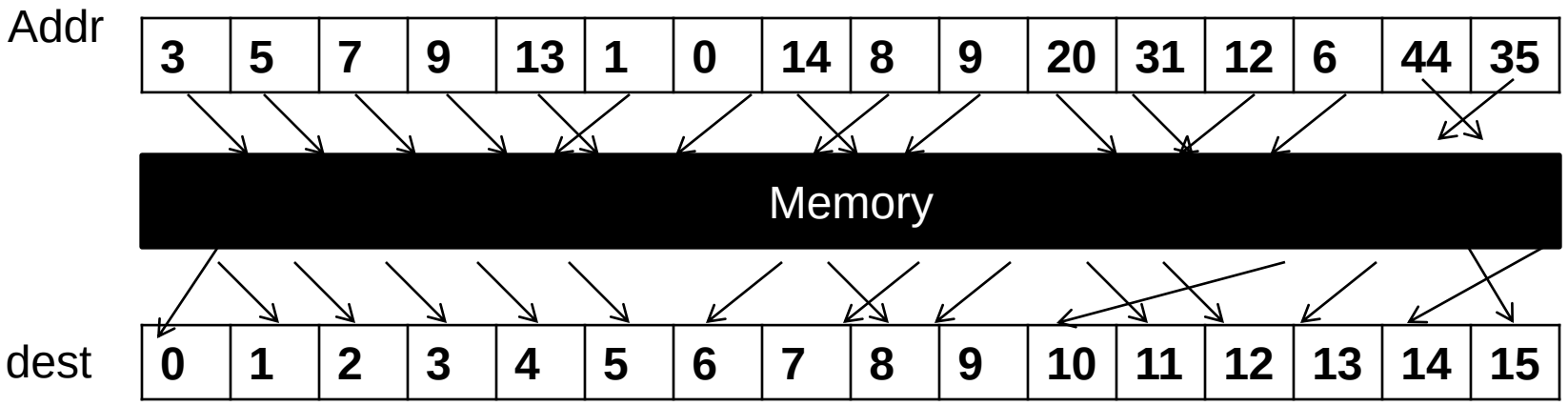
VPU (Vector Processor Units)



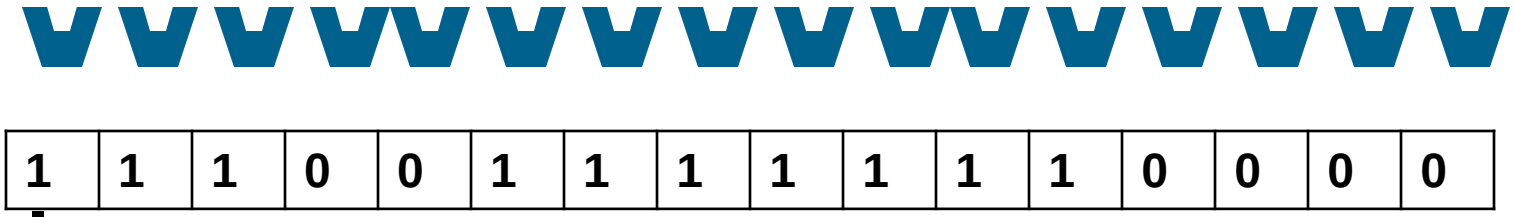
- 16-wide SIMD unit
  - 16 wide Single precision
  - 8-wide double precision
- Hardware scatter/gathering operations : 16 elements are loaded from or stored to up to 16 different addresses that are specified in another vector register.
- New instructions: fused multiply-add, and the standard logical operations, including instructions to extract non-byte-aligned
- Data can be replicated from L2 cache directly
- Free numeric type conversion and data
- replication while reading from memory
- Mask registers: predicated
- 3 source operands, one of them can come from L1 directly

Gather and Scatter Operation Support

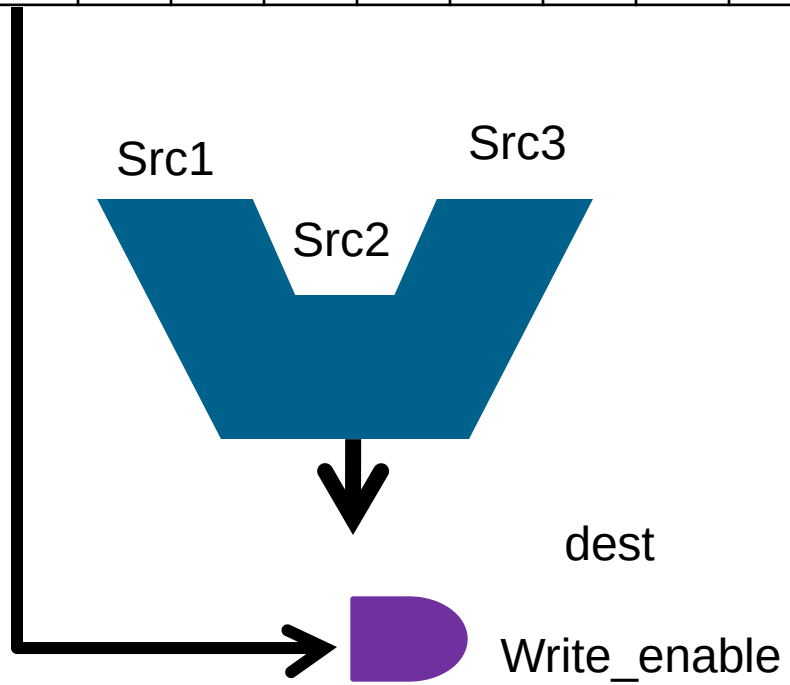
- Loads and stores from non-continuous addresses
- 16 data values can be loaded or stored from addresses in another vector register value.



Mask Bits

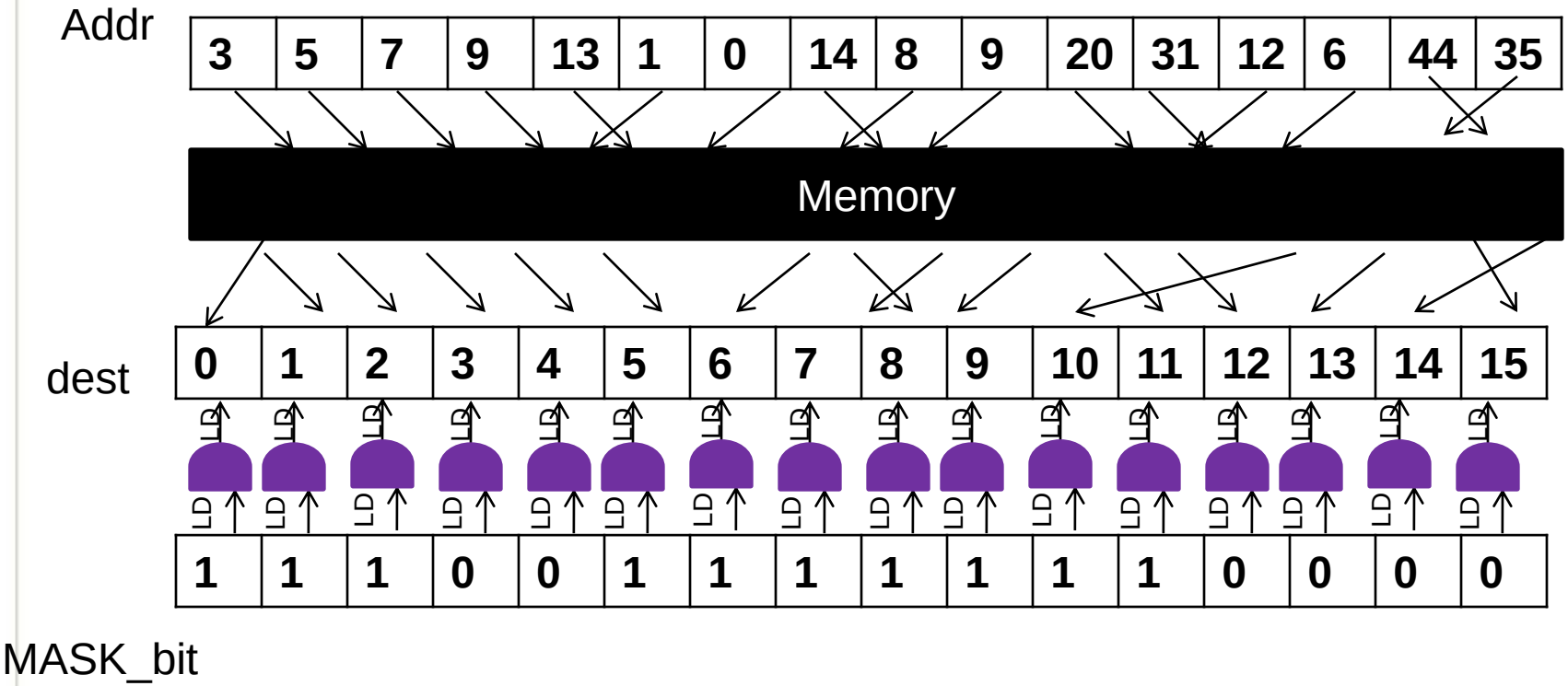


1	1	1	0	0	1	1	1	1	1	1	1	0	0	0	0



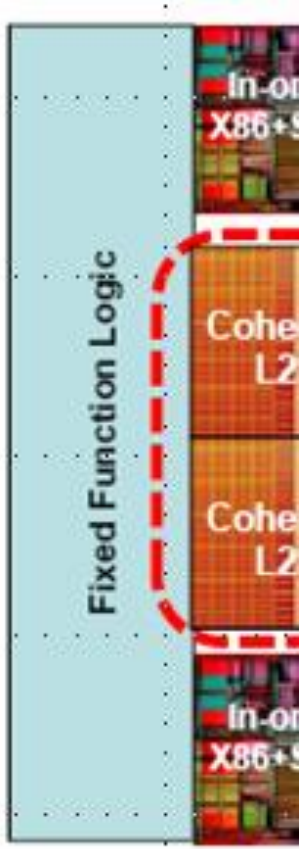
- All the operations participated the computation in vector units
- Mask bits decide write enable signal

Predication for Load/Store

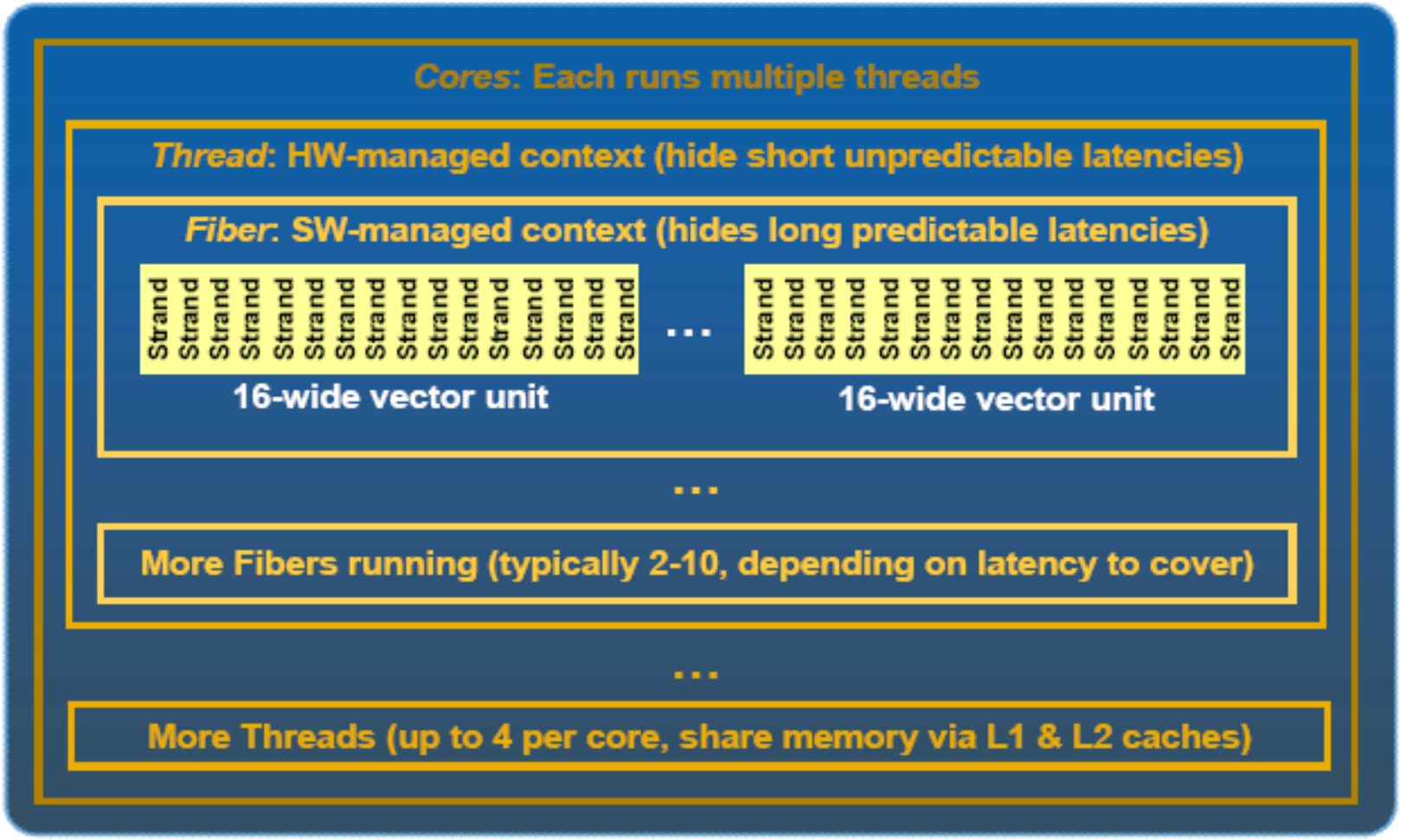


Fixed Functions

- Use FIFO for load balancing
- No rasterizations
- Texture filtering
  - 32KB texture cache per each core
  - Core passes commands through L2 cache
- Texture unites perform virtual to physical page translation



Larrabee Data Parallelism: Threads



http://www.ece.neu.edu/groups/nucar/GPGPU/GPGPU-2/Seiler.pdf



Data Parallelism Support

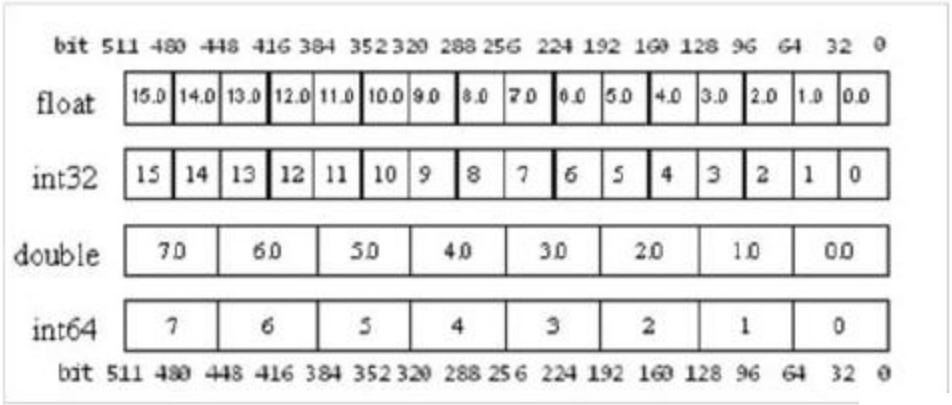
- Implementation hierarchy (your names may vary)
  - Strand: runs a data kernel in one (or more) VPU lane(s)
  - Fiber: SW-managed group of strands, like co-routines
  - Thread: HW-managed, swaps fibers to cover long latencies
- Core: runs multiple threads to cover short latencies

- Comparison to GPU data parallelism
  - Same mechanisms as used in GPUs, except…
  - Larrabee allows SW scheduling (except for HW threads)

http://www.ece.neu.edu/groups/nucar/GPGPU/GPGPU-2/Seiler.pdf

LRB new instruction sets

- New Data types



- Vector arithmetic, logical, an
- Vector mask generation
- Vector load/store

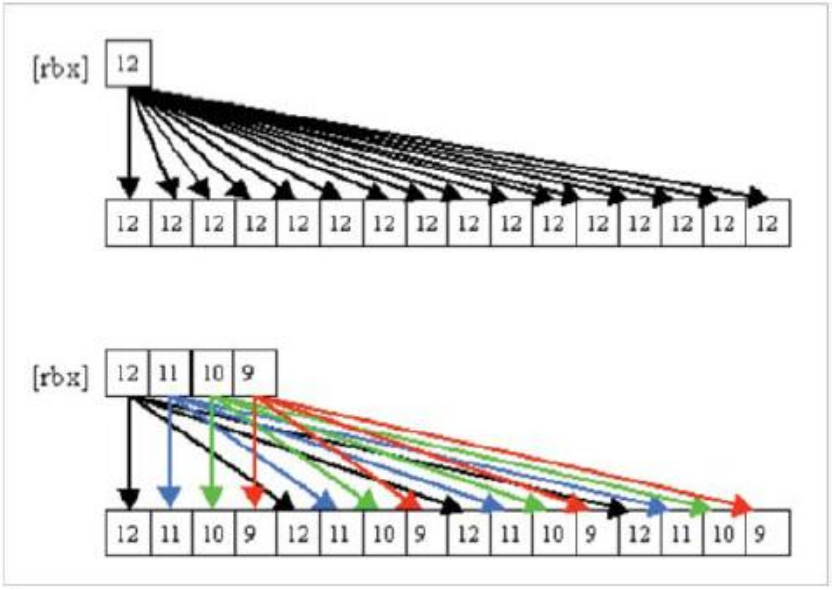


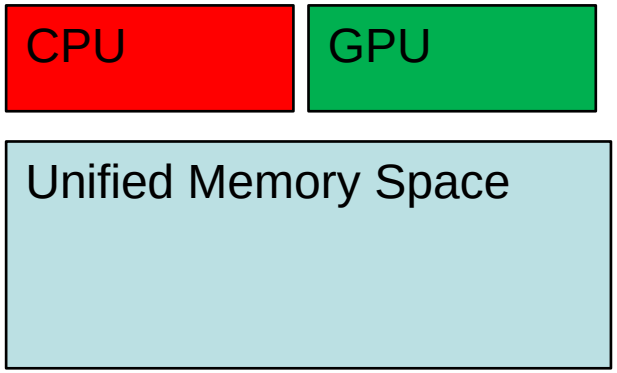
Figure 3: 1-to-16 {1to16} and 4-to-16 {4to16} broadcasts.



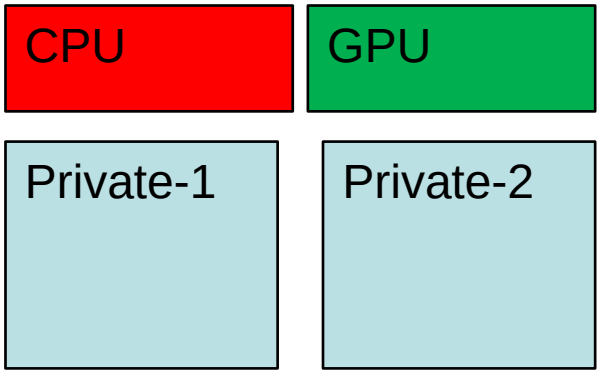
Is LRB failed?

- Knight Ferry
- Software Programming GPUs
  - → slow at Market

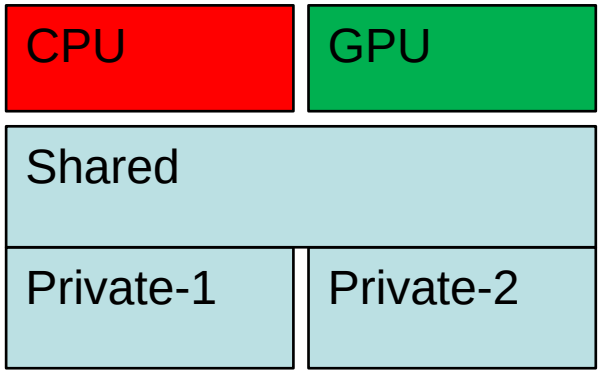
Programming Memory Spaces



Non-coherent: CPU 4.0, Rigel, Exochi



Cuda (3.0), OpenCL, Sandy Bridge, Cell



CPU+LRB

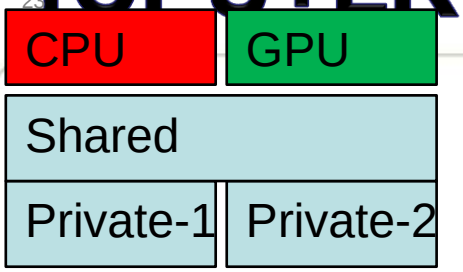
[CPU+NVIDIA GPU/OpenCL] Explicit Communication

- cudaMemcpy(, , , cudaMemcpyHostToDevice);
- cudaMemcpy(, , , cudaMemcpyDeviceToHost);

```
// Copy vectors from host memory to device memory
cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);
```

```
// Copy result from device memory to host memory
// h_C contains the result in host memory
cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
```

[CPU+LRB]



Vector addition example

```
int addTwoVectors(int* a, int* b, int* c)
{
   for (i = 1 to 64) {
      c[i] = a[i] + b[i]
  }
}

int someApp( …)
{
  int *a = malloc (..);
  int *b = malloc (..);
  int *c = malloc (..);
   for (i = 1 to 64) //initialize
       {a[i] = ; b[i] = ; c[i] = ;}
   addTwoVectors(a, b, c);
       …
}
```

Function should be executed on the LRB

```
__attribute(LRB) int addTwoVectors(shared
int* a, shared int* b, shared int* c)
{  acquireOwnership();
   for (i = 1 to 64) {
     c[i] = a[i] + b[i];
   }
   releaseOwnership();
}
int someApp(..)
{  // allocate in shared region
   shared int* a = sharedMalloc (..);
   shared int* b = sharedMalloc (..);
   shared int* c = sharedMalloc (..);
   for (i = 1 to 64) // initialize
       {a[i] = ; b[i] = ; c[i] = ;}
   releaseOwnership();
   // compiler converts into remote call
   addTwoVectors(a, b, c);
   acquireOwnership();
}
```

Thrust template {unified address space CUDA}

This code sample computes the sum of 100 random numbers on the GPU.

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/reduce.h>
#include <thrust/functional.h>
#include <cstdlib>

int main(void)
{
  // generate random data on the host
  thrust::host_vector<int> h_vec(100);
  thrust::generate(h_vec.begin(), h_vec.end(), rand);

  // transfer to device and compute sum
  thrust::device_vector<int> d_vec = h_vec;
  int x = thrust::reduce(d_vec.begin(), d_vec.end(), 0, thrust::plus<int>());
  return 0;
}
```