Debunking the GPU CPU Myth

Lee et al

What are CPUs tailored for/not talored for?

- Many types of applications
- Complex cores for general purpose
- Fast response time
- Fast synchronization primitives
- Shuffle/swizzle SIMD instructions

- High area
- High power
- Not good for thread switching

What are GPUs tailored for/not tailored for?

- Single application – graphics
- Simple cores, but many of them
- Latency tolerant
- Easy thread-switching (hardware support)
- Gather/scatter

- Small caches
- Too many challenges if irregular
- Challenging if working set don't fit

Is the paper biased towards Intel? Are the results not believable?

- Num PE 4:30, so no more than 8 to be expected
- Freqy 3.2 :1.3 so speedup of 8 gets lowed by 2X
- Peak Scalar flops 25:116 i.e. approx 4.5X only
- Peak Single precision SIMD flops only 3X or 9X if FMA
- Peak Double precision SIMD flops only 1.5X
- Peak BW – 32:141 only 4.7X
- Bytes per flop is only 1.6X considering FMA
- CPU has large caches
- CPU has fast synch

Perf Analysis

- Mem BW ratio is 4.7X GPU
- Memory Bandwidth limited SAXPY and LBM – 5X
- SpMV – i7 cache fits vector; so only 1.9X
- GPU BW requirement for SpMV is 2.5X i7's requirement

- COMPUTE FLOPS ratio is 3-6
- SGEMM, MC, Conv, FFT, Bilat get 2.8-4X

PERF ANALYSIS CONTD

- CACHES bigger in i7
- SORT working set fits in cache; SORT is faster in CPU
- SpMV working set fits in cache. SpMV gives only 1.9X
- Instead of the 5X possible speedup

- GATHER-SCATTER
- GJK and RC need gather support; poor perf on i7
- I7 needs 20 instructions for compiler-generated gather
- I7 needs 13 instructions for assembly optimized gather

PERF ANALYSIS CONTD

- Reduction and Synchronization
- Hist makes atomic updates
- Solv needs barrier synchronization
- I7's cache coherency helps

- FIXED FUNCTION
- Bilat and MC – transcendental operations
- GJK – benefits from texture lookup on GPU

HARDWARE RECOMMENDATIONS

- High Compute flops needed – more PEs or higher SIMD width
- High Mem BW needed - pin and power issues -3D stacking
- Large caches – match working set and on-die storage
- Gather-scatter – would be nice to gather all elements into SIMD
- register in the same amount of time to load one cache line.
- Need large # cache ports or multi-banking
- Shuffle logic
- Synchronization and Cache coherence – atomic operations, red
- Fixed function units – transcendental, CRC, encryption/decrypti
- Texture sampling units

Intel Gather-Instruction

Inputs	Outputs
BASE_ADDR: Base Address	*ZMM*: *Float32* output vector
VINDEX: Doubleword Index Vector	*K*: Write Mask
SCALE: Scaling factor	
K: Write Mask	

Intel Gather-Instruction

```
#zmm0 – Index register

    #r8    - Base Address

    #k1    - Mask register

    #zmm6 – Output register

    #4     - Scale

    ..L10:

            vgatherdps (%r8,%zmm0,4), %zmm6{%k1}

            jkzd      ..L9, %k1

            vgatherdps (%r8,%zmm0,4), %zmm6{%k1}

            jknzd     ..L10, %k1

    ..L9:
```

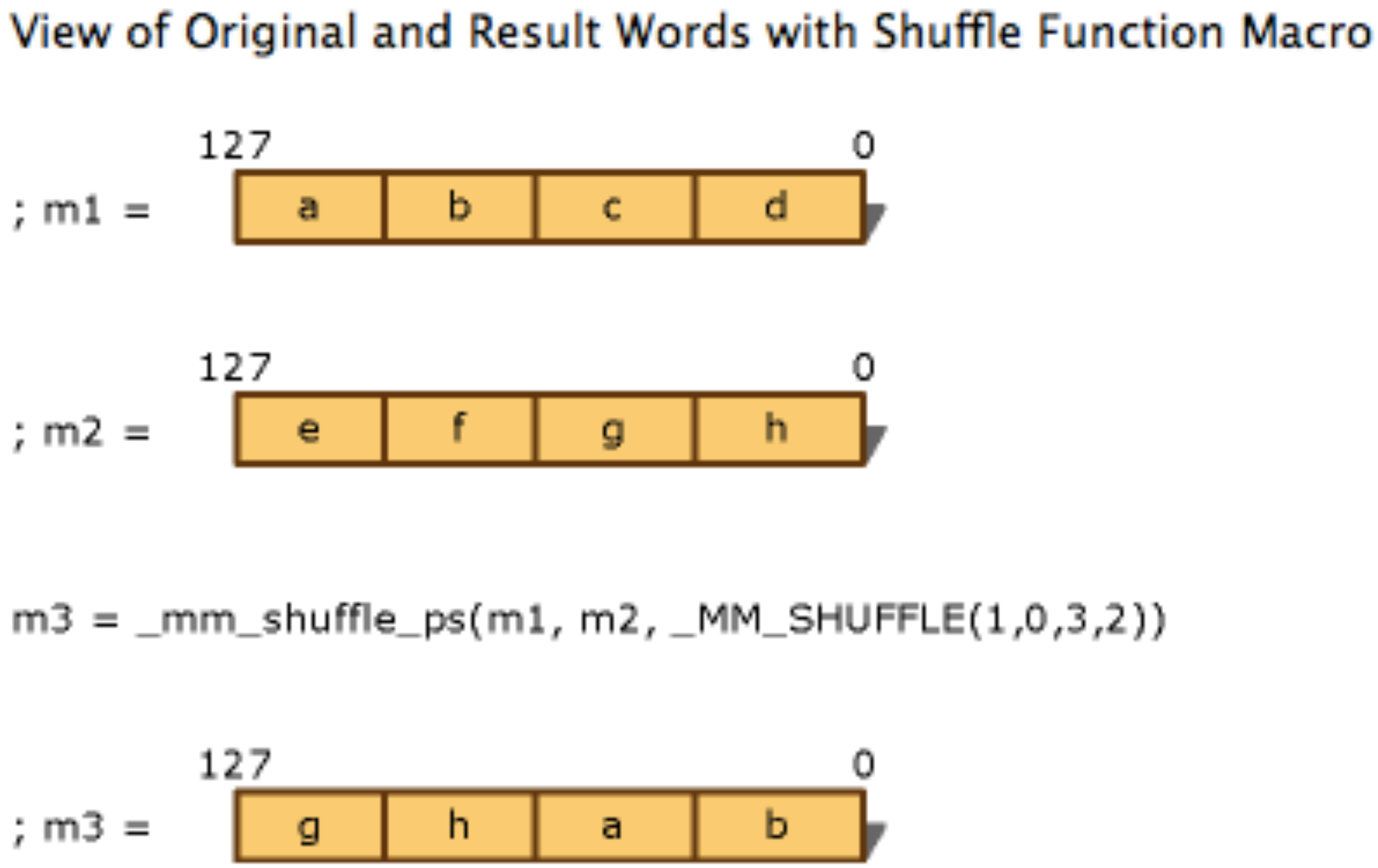
Intel Scatter-Instruction

```
#zmm0 – Index register
      #r12  - Base Address
      #k3   - Mask register
      #zmm6 – Input register
      #4    - Scale
      ..L14:
            vscatterdps %zmm7, (%r12,%zmm6,4){%k3}
            jkzd      ..L13, %k3
            vscatterdps %zmm7, (%r12,%zmm6,4){%k3}
            jknzd     ..L14, %k3
      ..L13:
```

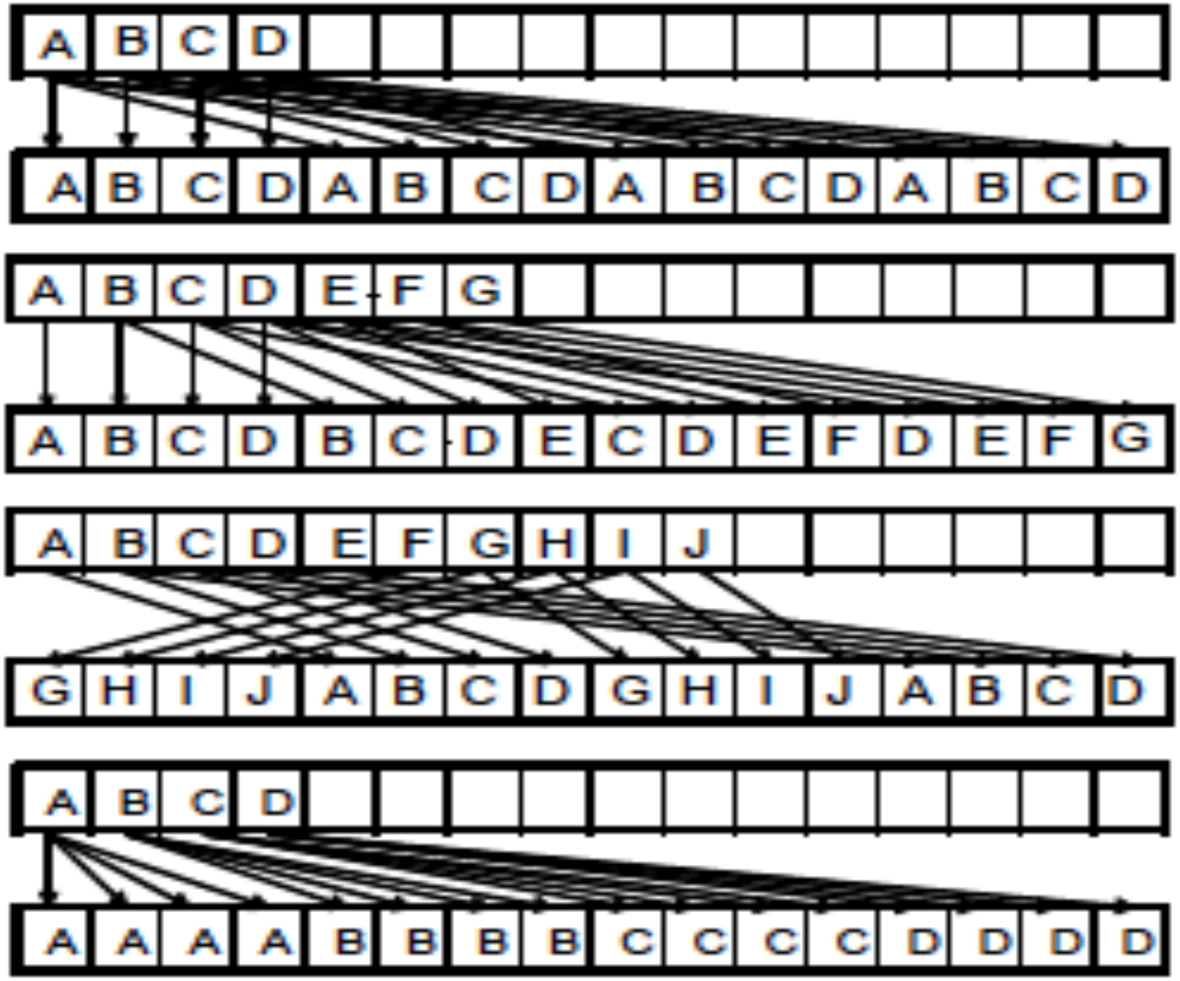
GATHER-SCATTER

- Useful for Sparse Matrices and Arrays
- Gather is eqvt to Load Vector Indexed
- Scatter is eqvt to Store Vector Indexed
- Gather loads data items from sparse arrays
- Scatter writes data items to sparse arrays
- Use masks
- Gather scatter can be implemented using linked lists

SHUFFLE


View of Original and Result Words with Shuffle Function Macro
127
0
; m1 =
a
b
c
d
127
0
; m2 =
e
f
g
h
m3 = _mm_shuffle_ps(m1, m2, _MM_SHUFFLE(1,0,3,2))
127
0
; m3 =
g
h
a
b

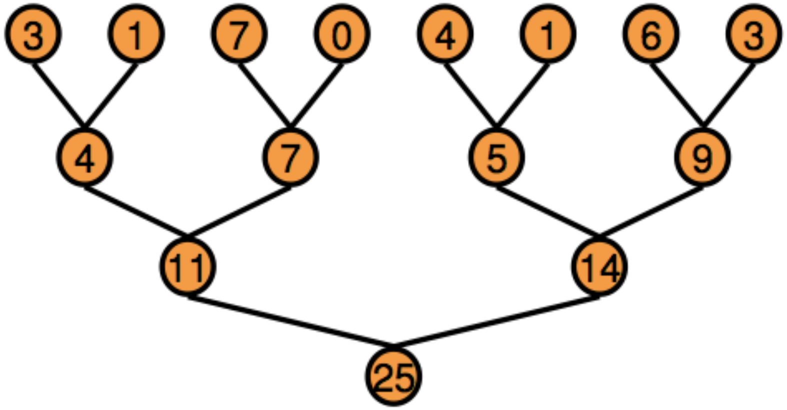
Swizzle/SPLAT



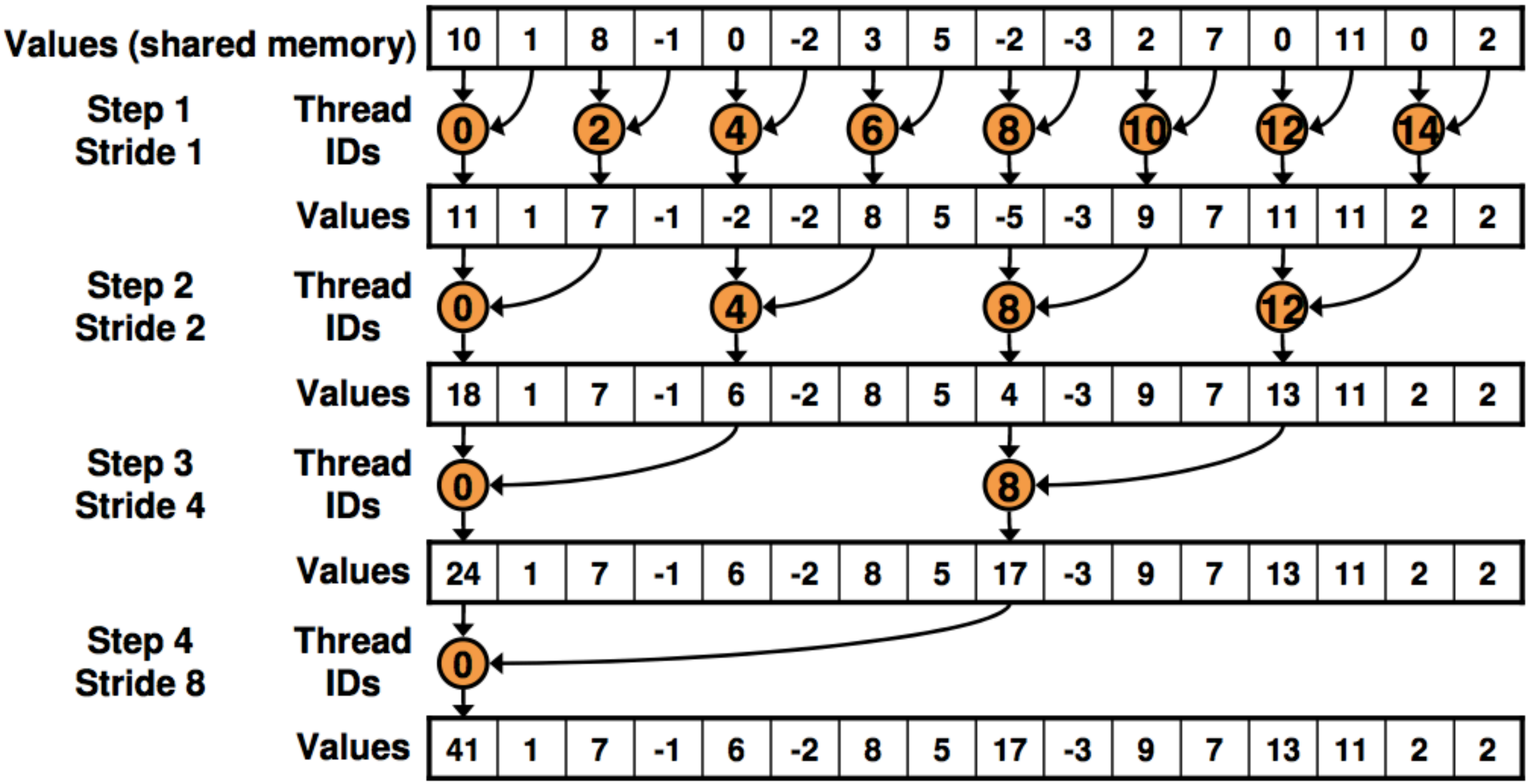
REDUCTION

Parallel Reduction

- **Tree-based approach used within each thread block**



Parallel Reduction: Interleaved Addressing



What is VOXEL?

- PIXEL is Picture Element
- VOXEL is VOlume ELement

Atomic Operations

- Compare and Swap
- Fetch and add
- Lock and Increment
- Test and Set
- **compare-and-swap** (CAS) used in multithreading

to achieve synchronization. It compares the contents of a memory location to a given value and, only if they are the same, modifies the contents of that memory location to a given new value