Atomic Vector Operations on Chip Multiprocessors

Microprocessor Technology Lab, Intel Corp.
Sanjeev Kunar, Dachyun Kim,
Mikhail Smelyanskiy, Yen-Kuang Chen,
Jatin Chhugani, Christopher J. Hughes,
Changkyu Kim, Victor W. Lee, Anthony D. Nguyen
Motivation – SIMD Deficiencies

- Control flow
  - Masked SIMD instructions
- Irregular data access
  - Rearrange data
  - Gather/Scatter Unit (GSU) in addition to Load/Store Unit (LSU)
- Atomic operations
  - Lock acquires, parallel reductions
  - LoadLinked/StoreConditional (LL, SC) inst.
Architecture support

• Small extension to exiting hw
• Best-effort model
  - Even loads can fail!
  - Semantics match LL-SC model
  - Output mask indicating success/failure
• It's GLSC!
  - Gather Linked – Scatter Conditional
• Aliasing issue
  - Conventional G/S - undefined
A: Using GLSC to perform the reduction directly

```c
for( i = 0; i < numPixels; i += SIMD_WIDTH) {
    // Load the next SIMD_WIDTH inputs into Vinput
    vload Vinput, &Minput[i];
    // Compute the bins
    vmod Vbins, Vinput, numBins;
    FtoDo = ALL_ONES;
    do {
        // Do remaining elements specified by FtoDo
        Ftmp = FtoDo;
        vgatherlink Ftmp, Vtmp, Mbins, Vbins, Ftmp;
        vinc Vtmp, Vtmp, Ftmp; // Increment bins
        vscattercond Ftmp, Vtmp, Mbins, Vbins, Ftmp;
        // Record elements that processed successfully
        FtoDo ^= Ftmp;
    } while (FtoDo != 0);
}
```
ISA Extension – 2 new instructions

- **vgatherlink Fdst, Vdst, base, Vindx, Fsrc**
  - Loads data
  - Create vector
  - Reserves mem. locations

- **vscattercond Fdst, Vsrc, base, Vindx, Fsrc**
  - Unpacks vector
  - Stores data to reserved mem locations

- Also usefull for vectorized locking
  - Ala. POSIX semop()
Failures in detail

- Linking is on L1 cacheline level
- GL may fail (as opposed to LL):
  - Another thread has linked the cache line
  - Linking would evict another linked line (associativity)
  - The latency of one element is significantly higher
- Fails allow shorter hold times
- No rearranging to force uniqueness
- Can handle pagefaults
HW implementation

- Extend existing GSU
  - LL instead of loads
  - SC instead of stores
- Extend L1D$
  - New GLSC entry (each line)
    - Valid bit, hardware thread id (vcpu id)
    - Less than 1% of data size
HW implementation Example
Modeled system

- In-order
- 1-4 cores, 1-4 threads on each
- SIMD width 1-16
- Hw coherent L1
- GSU issues at most 1 request per cycle
- LSU has higher priority over GSU
- BASE:
  - Uses LL/SC
Benchmarks

- Parallelized and vectorized by authors
- Use atomic RMW, datasets: A, B
- 8 benchmarks
  - Grid-based Collision Detection (GBC)
  - Forward Triangular Solve (FS)
  - Game Physics Solver (GPS)
  - Histogram for Image Processing (HIP)
  - Surface Extraction and Marching Cubes (SMC)
  - Maxflow Push (MFP)
  - Transpose Matrix-Vector Multiply (TMS)
Results: SIMD Width

(a) Synchronization Time

(b) SIMD Efficiency

Benchmarks: A, B, GBC, FS, GPS, HIP, SMC, MFP, TMS

Percentage of Execution Time

16-wide SIMD
4-wide SIMD
Results: 4-wide SIMD
## Analysis

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dataset</th>
<th>Reduction with GLSC on 4x4</th>
<th>GLSC failure rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Instructions</td>
<td>Memory Stalls</td>
</tr>
<tr>
<td>GBC</td>
<td>A</td>
<td>15.94 %</td>
<td>21.80 %</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>13.81 %</td>
<td>17.36 %</td>
</tr>
<tr>
<td>FS</td>
<td>A</td>
<td>75.43 %</td>
<td>59.19 %</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>60.53 %</td>
<td>21.86 %</td>
</tr>
<tr>
<td>GPS</td>
<td>A</td>
<td>18.79 %</td>
<td>-3.83 %</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>20.76 %</td>
<td>-5.46 %</td>
</tr>
<tr>
<td>HIP</td>
<td>A</td>
<td>28.12 %</td>
<td>n/a %</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>39.85 %</td>
<td>n/a %</td>
</tr>
<tr>
<td>SMC</td>
<td>A</td>
<td>29.96 %</td>
<td>45.48 %</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>33.59 %</td>
<td>49.93 %</td>
</tr>
<tr>
<td>MFP</td>
<td>A</td>
<td>18.85 %</td>
<td>12.46 %</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>17.66 %</td>
<td>15.08 %</td>
</tr>
<tr>
<td>TMS</td>
<td>A</td>
<td>47.10 %</td>
<td>56.02 %</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>52.28 %</td>
<td>37.62 %</td>
</tr>
</tbody>
</table>

The **Instructions** column shows the reduction in the number of instructions for GLSC compared to Base. The **Memory Stalls** column shows the reduction in the number of stall cycles due to memory accesses for GLSC compared to Base. This number is *n/a* for HIP because its implementation with GLSC and Base are different. The **L1 Accesses** column shows two numbers for each row. The second number shows the percentage of L1 accesses due to atomic operations (i.e., GLSC). The first number shows the percentage reduction in the number of L1 accesses due to atomic operations because of cache line reuse in the gather-scatter unit. The last two columns show the percentage of atomic operations that fail due to aliasing or collisions between threads.
Analysis – Benefits

- Instruction reduction
  - 33.8% on average

- L1$ miss latency overlap
  - 2 or more instructions miss, wait for same cache lines
  - 24.4% fewer memory stalls
  - Acts like prefetch, actual explicit prefetch not as good

- Fewer L1$ accesses
  - GSU
  - Mostly HIP, and FS (16% and 17%)
Analysis – Problems

- Cache associativity smaller than SIMD width
  - Not on test system
- Element aliasing
  - The only problem in 1x1 configuration
  - Mostly HIP and GBC
- Competing threads running atomic ops
  - 4x4 vs 1x1 (less than 0.1%)
Microbenchmarks – What helps?

- **Scenario A** (overlapping cache misses)
  - Every SIMD element on new cacheline
    - No aliasing, no cacheline reuse
    - Multiple threads, cacheline stealing
- **Scenario B** (fewer inst, fewer L1$ accesses)
  - SIMD elements on the same cache line
  - No aliasing
  - Single thread
Microbenchmarks – What helps?

• Scenario C (fewer instructions)
  – Like B, but elements on different lines

• Scenario D (no advantages)
  – Like B
  – Every SIMD element is the same address (all alias)
Microbenchmark results

![Bar chart showing relative execution times for different scenarios and SIMD widths.]

- **SIMD width 4:**
  - Scenario A: 3
  - Scenario B: 2
  - Scenario C: 2
  - Scenario D: 1

- **SIMD width 16:**
  - Scenario A: 4
  - Scenario B: 3
  - Scenario C: 3
  - Scenario D: 1
Conclusion

• 1-wide SIMD
  – Only one op in SIMD succeeds
  – GLSC won't be worse than current approach

• 16-wide SIMD
  – 103% improvement on average (compared to 1wide)

• Significant performance benefits for little hw cost
  – 76% avg. performance improvement in single thread
  – 54% avg. performance improvement in 4x4
Thank you for your attention!