Efficient SIMD Vectorization for Hashing in OpenCL

Tobias Behrens[1], Viktor Rosenfeld[1], Jonas Traub[2], Sebastian Breß[1,2], Volker Markl[1,2]

German Research Center for Artificial Intelligence

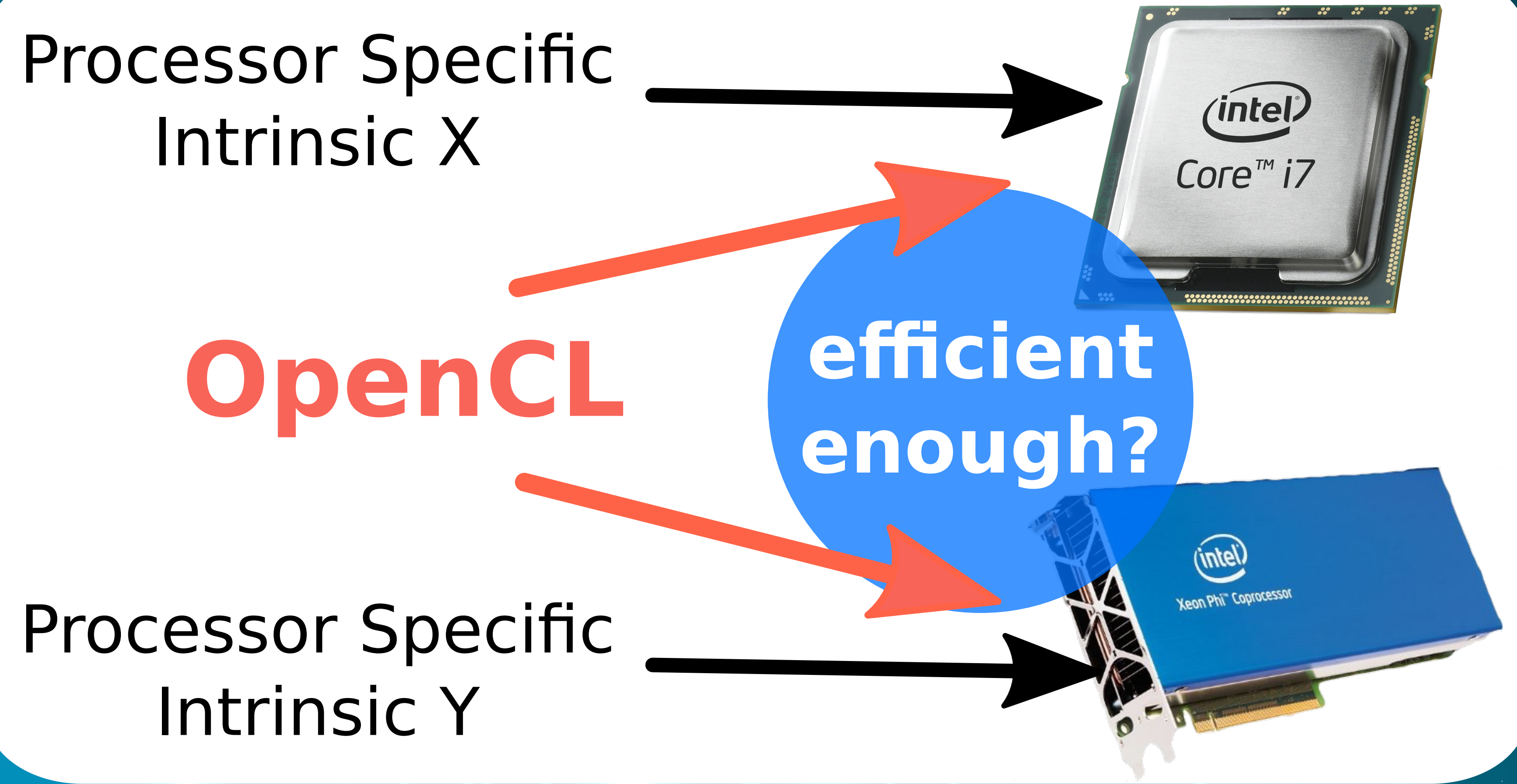
[1]firstname.lastname@dfki.de
[2]firstname.lastname@tu-berlin.de

Abstract

Hashing is at the core of many efficient database operators such as hash-based joins and aggregations.

Significant speedup was shown for vectorized hash table operations using processor specific low-level intrinsics.

We present portable and vectorized hashing primitives using the parallel programming framework OpenCL.



Processor Specific Intrinsic X

OpenCL

efficient enough?

Processor Specific Intrinsic Y

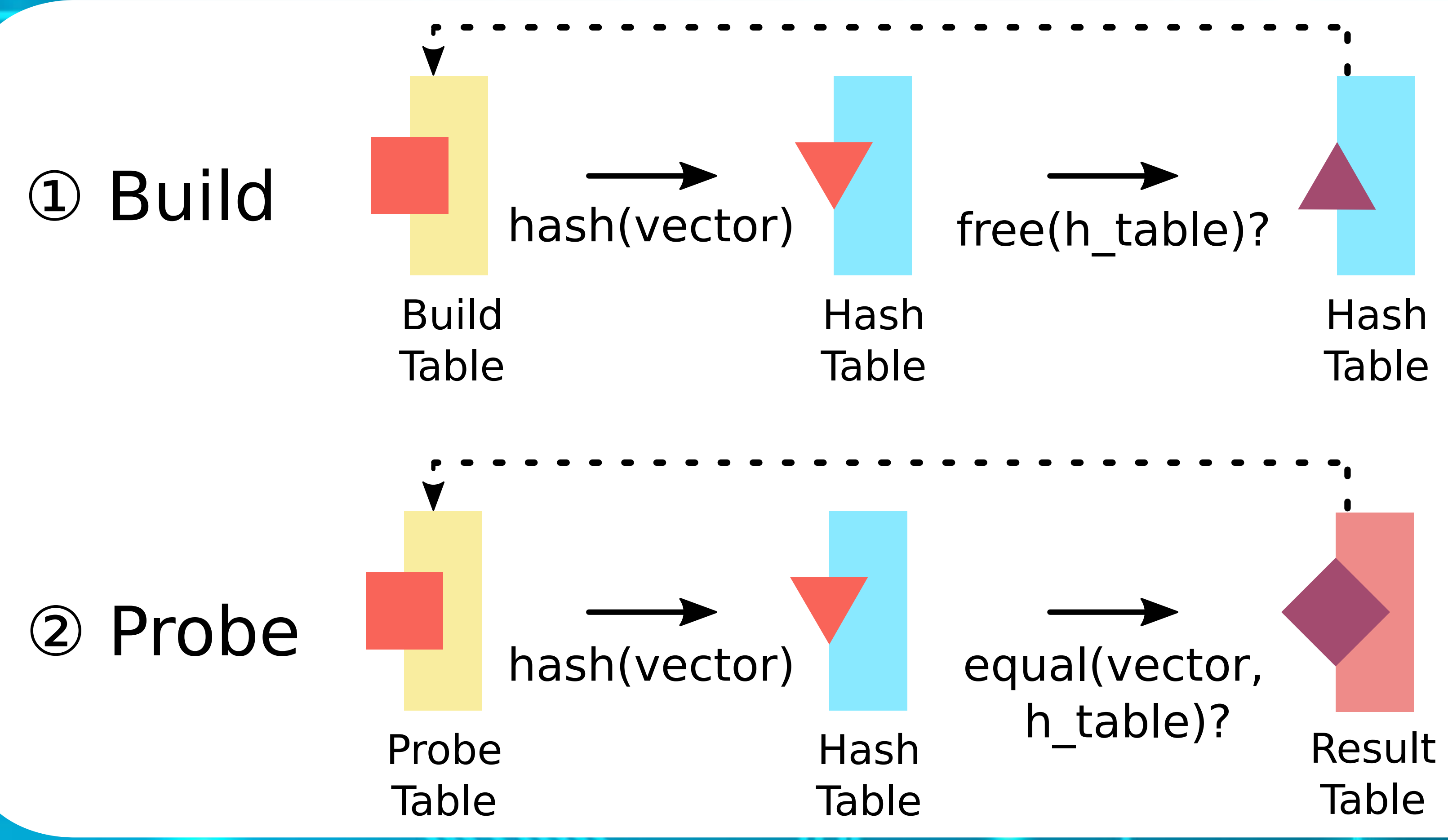
Vectorized Data Movement Primitives

Selective Load – Copies data from contiguous memory into selectable SIMD lanes. (Probe / Build Table)

Selective Store – Copies data from selectable SIMD lanes to contiguous memory. (Result Table)

Gather – Copies data from discontiguous memory into SIMD lanes. (Hash Table)

Scatter – Copies data from SIMD lanes into discontiguous memory. (Hash Table)



① Build

Build Table → hash(vector) → Hash Table → free(h_table)? → Hash Table

② Probe

Probe Table → hash(vector) → Hash Table → equal(vector, h_table)? → Result Table

Gather OpenCL

```
// tables: Bucket *table, uint 8 mask
// Output: uint8 vector
vector.s0 = table[mask.s0].key;
vector.s1 = table[mask.s1].key;
//   ... up to vector.s7 = table[mask.s7].key
```

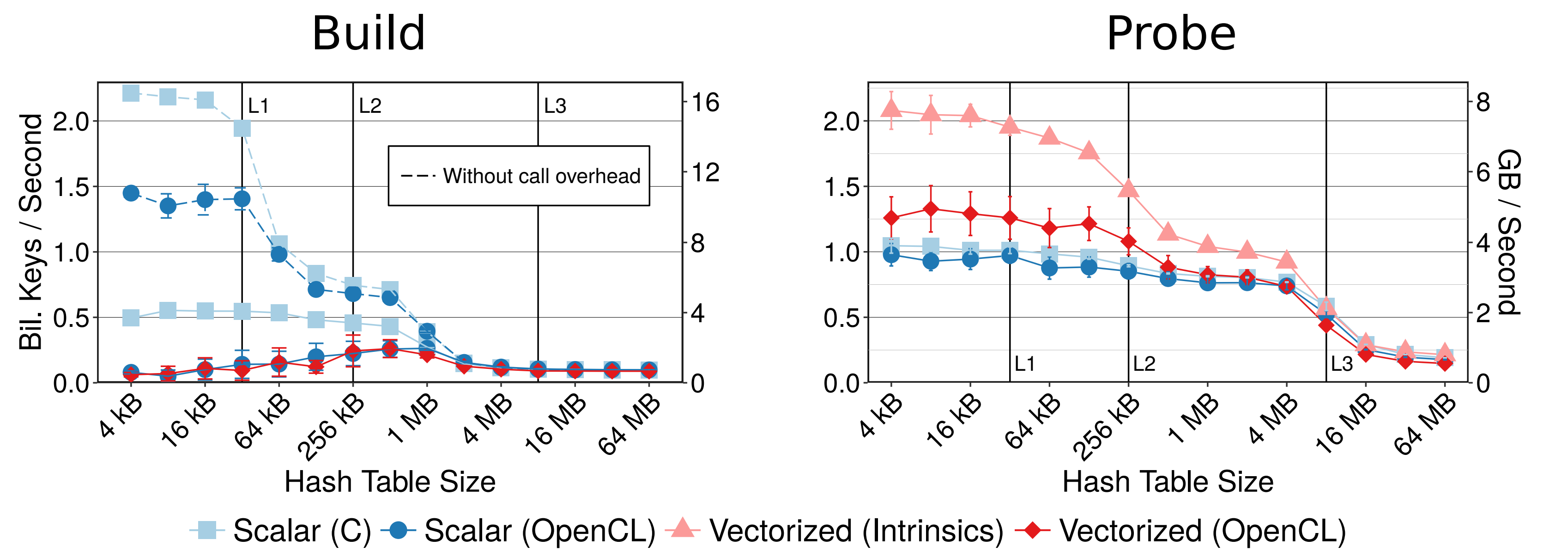
vs.

Gather Intel Intrinsics

```
// Inputs: uint64_t* table, __128i index
// Output: __m256i res
__m128i index_R = _mm_shuffle_epi32(index, _MM_SHUFFLE(1, 0, 3, 2));
__m128i i12 = _mm_cvtepi32_epi64(index);
__m128i i34 = _mm_cvtepi32_epi64(index_R);
size_t i1 = _mm_cvtsi128_si64(i12);
size_t i3 = _mm_cvtsi128_si64(i34);
__m128i d1 = _mm_loadl_epi64((__m128i *)&table[i1]);
__m128i d3 = _mm_loadl_epi64((__m128i *)&table[i3]);
i12 = _mm_srli_si128(i12, 8);
i34 = _mm_srli_si128(i34, 8);
size_t i2 = _mm_cvtsi128_si64(i12);
size_t i4 = _mm_cvtsi128_si64(i34);
__m128i d2 = _mm_loadl_epi64((__m128i *)&table[i2]);
__m128i d4 = _mm_loadl_epi64((__m128i *)&table[i4]);
__m256i d12 = _mm256_castsi128_si256(_mm_unpacklo_epi64(d1, d2));
__m256i d34 = _mm256_castsi128_si256(_mm_unpacklo_epi64(d3, d4));
__m256i res = _mm256_permute2x128_si256(d12, d34, _MM_SHUFFLE(0, 2, 0, 0));
```

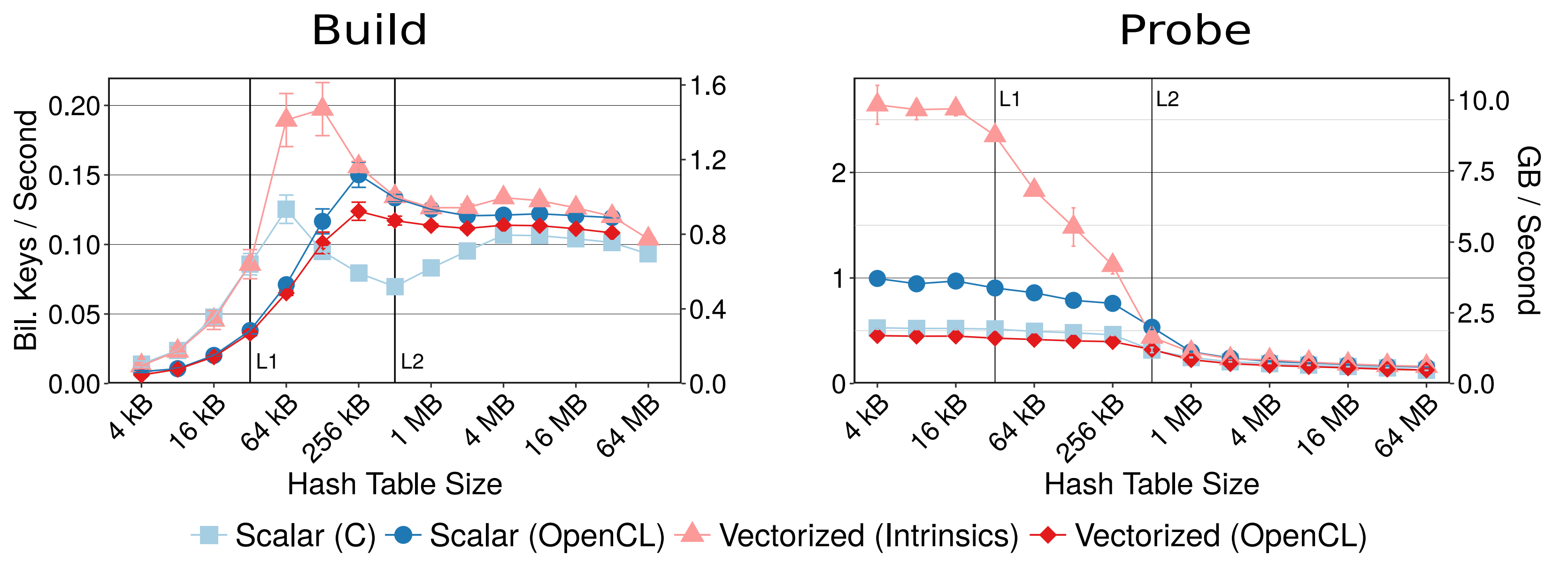
Portable and Maintainable Code

Performance on Xeon CPU



Build is overhead dominated, OpenCL-based probe outperforms scalar implementation.

Performance on Xeon Phi



Intrinsic-based implementation outperforms OpenCL-based on Xeon Phi.

Take Home

OpenCL reduces code complexity and ensures portability of vectorized primitives.

OpenCL-based vectorized hashing outperforms scalar hashing on Xeon CPU.

Processor specific intrinsics are still faster, especially on Xeon Phi.

Open Source Repository

github.com/TU-Berlin-DIMA/OpenCL-SIMD-hashing

Funding Acknowledgements

This work was funded by the EU projects SAGE (671500) and E2Data (780245), DFG Stratosphere (606902), and the German Ministry for Education and Research as BBDC (01IS14013A) and Software Campus (01IS12056).