Exercise Session 9

Data Processing on Modern Hardware
263-3502-00L — Fall Semester 2012

Cagri Balkesen
cagri.balkesen@inf.ethz.ch

Department of Computer Science ETH Zurich, Switzerland

15 November 2012

4.2 Warm-up Exercise

- Task : implement SIMD-accelerated version `compareandcount.c`
  - Array is filled with 100 mio numbers $\in [0, 99]$
  - Counts how many values > 42
- Many different versions possible
- Three solutions presented:
  - (A) 4 elements at a time
  - (B) 16 elements at a time (version 1)
  - (C) 16 elements at a time (version 2)
- Experiments on : Intel(R) Core(TM) i5 CPU 750@2.67GHz
- Solution $\rightarrow$ run : `sh ./compareandcount.sh`

(A) 4 Elements at a Time

```
/* init registers */
const __m128i cmpMask = \
_mm_set_epi32(42,42,42,42);
__m128i accumulator = \
_mm_setzero_si128();

/* 4 masks → 4-way loop unrolling */
const __m128i shuffmask1 = \
_mm_set_epi8 (0xff, 0xff, 0xff, 3,...);
const __m128i shuffmask2 = \
_mm_set_epi8 (0xff, 0xff, 0xff, 7,...);
const __m128i shuffmask3 = \
_mm_set_epi8 (0xff, 0xff, 0xff, 11,...);
const __m128i shuffmask4 = \
_mm_set_epi8 (0xff, 0xff, 0xff, 15,...);
```

- serial : 95627 usec
- parallel-4 : 15100 usec
- speed-up : **6.33**

```
for (i = 0; i < ARRAYSIZE/16; i++) {
  next16 =_mm_loadu_si128((__m128i *) \
  (numberarray+i*16));
  next4 = \
  _mm_shuffle_epi8(next16, shuffmask1);
  next4 = \
  _mm_cmpgt_epi32(next4, cmpMask);
  accumulator = \
  _mm_add_epi32(accumulator, next4);
  /* repeat the above steps 3 times */
  next4 = \
  _mm_shuffle_epi8(next16, shuffmask2);
  ...
}

parallelcnt = \
_mm_extract_epi32(accumulator,0) + \
_mm_extract_epi32(accumulator,1) + \
_mm_extract_epi32(accumulator,2) + \
_mm_extract_epi32(accumulator,3);

parallelcnt *= -1;
```

(B) 16 Elements at a Time

```
/* init registers */
const __m128i cmpMask = \
_mm_set_epi8(42,42,...,42,42);
__m128i accumulator = \
_mm_setzero_si128();
```

- serial : 95627 usec
- parallel-4 : 10141 usec
- speed-up : **9.43**
- speed-up of a student with similar approach: **10.847**

```
for (i = 0; i < ARRAYSIZE/16; i++) {
  next16 =_mm_loadu_si128((__m128i *) \
  (numberarray+i*16));
  next16 = \
  _mm_cmpgt_epi8(next16, cmpMask);
  accumulator = \
  _mm_add_epi8(accumulator, next16);

  /* prevent overflow :  8-bit!  */
  if((i+1)%128==0) {
    parallelcnt += \
    (signed char)
    _mm_extract_epi8(accumulator,0) + \
    ...
    (signed char)
    _mm_extract_epi8(accumulator,15);
    accumulator = _mm_setzero_si128();
  }
}
if((i+1)%128!=0) {
  /* add remaining counts */
  parallelcnt += ...;
}
```

(C) 16 Elements at a Time

```
/* init registers */
const __m128i cmpMask = \
_mm_set_epi8(42,42,...,42,42);
__m128i accumulator = \
_mm_setzero_si128();
__m128i zero = \
_mm_setzero_si128();
```

- serial : 95627 usec
- parallel-4 : 9446 usec
- speed-up : **10.12**

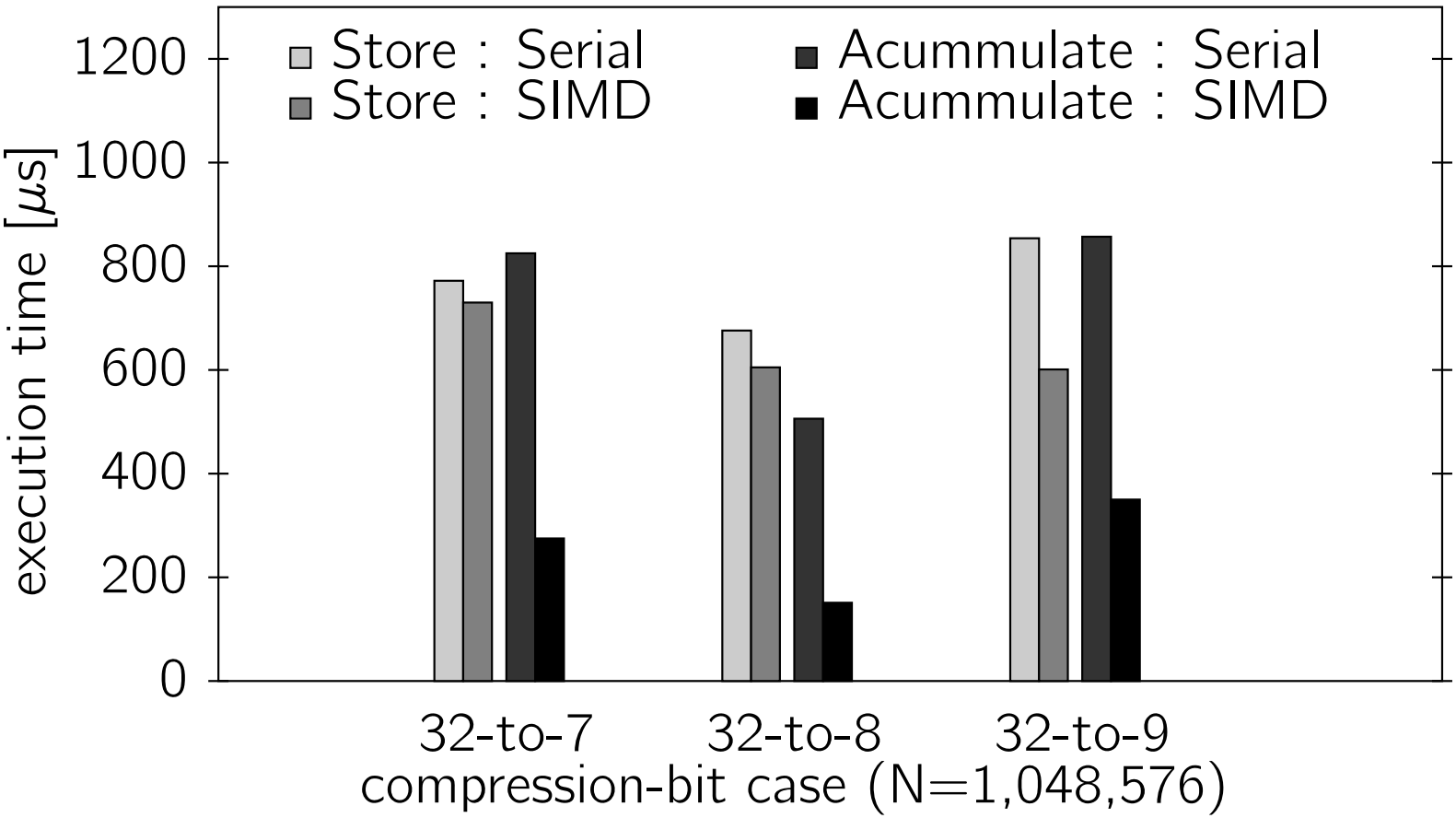
```
/* _mm_sad_epu8(a,b) :  computes
absolute difference of 16 integers from
a and b.  Sums upper 8 differences and
lower 8 differences and packs resulting
two unsigned 16-bit integers into the
upper and lower 64-bit elements.  */

for (i = 0; i < ARRAYSIZE/16; i++) {
  next16 =_mm_loadu_si128((__m128i *) \
  (numberarray+i*16));
  next16 = \
  _mm_cmpgt_epi8(next16, cmpMask);
  next16 = \
  _mm_abs_epi8(next16);
  next16 = \
  _mm_sad_epu8(next16,zero);
  accumulator = \
  _mm_add_epi64(accumulator, next16);
}
parallelcnt = \
_mm_extract_epi64(accumulator,0) + \
_mm_extract_epi64(accumulator,1);
```

4.3 Compression in Column Stores

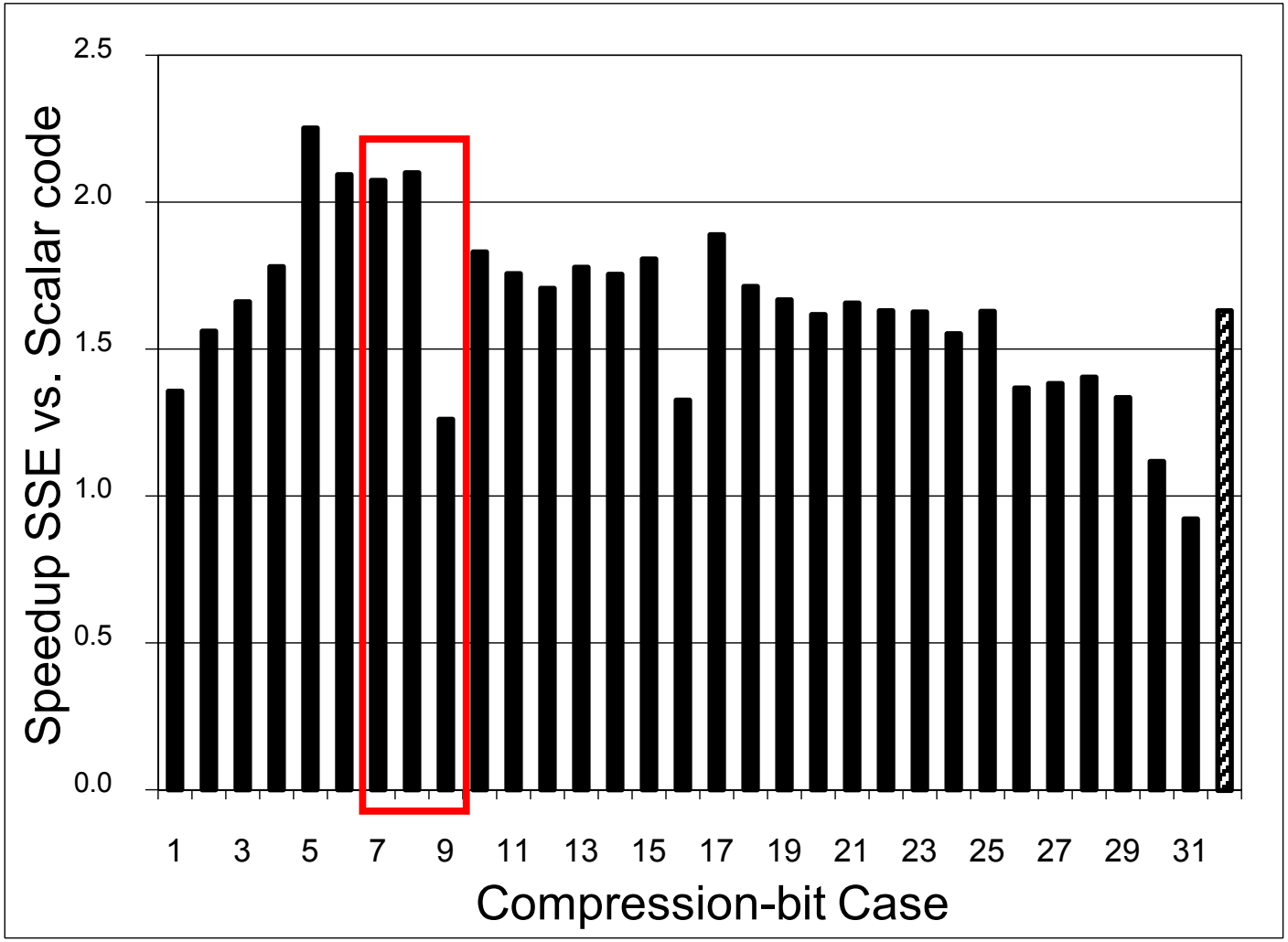
- Task : Implement the following decompression functions using SSE:
  - `SIMD_decompress8to32(...)`
  - `SIMD_decompress9to32(...)`
  - `SIMD_decompress7to32(...)`
- Optional : Consider not storing the values back to memory, i.e., compute some aggregate instead
- Experiments on : Intel(R) Core(TM) i5 CPU 750@2.67GHz
- Solution → run : `sh ./compression.sh`
  - 8-to-32 decompression → trivial
  - 9-to-32 decompression → showed in lecture
  - 7-to-32 decompression → explained now (blackboard)

Measurements : Absolute Execution Time



- Speedup of using SSE regarding different compression types becomes visible when we do not store back to memory.

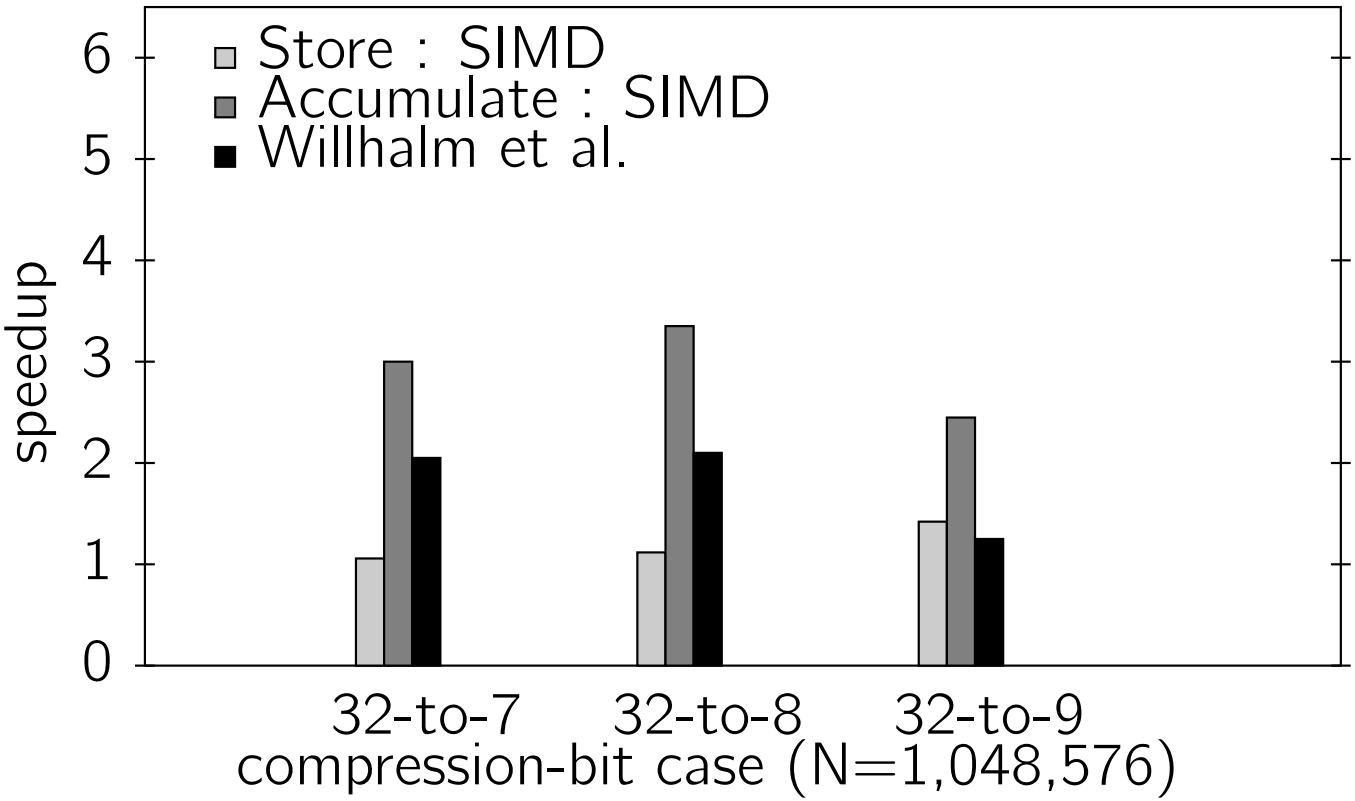
Comparison with Literature



Source: Willhalm *et al.* SIMD-Scan: Ultra Fast in-Memory Table Scan using on-Chip Vector Processing Units. *VLDB 2009*.

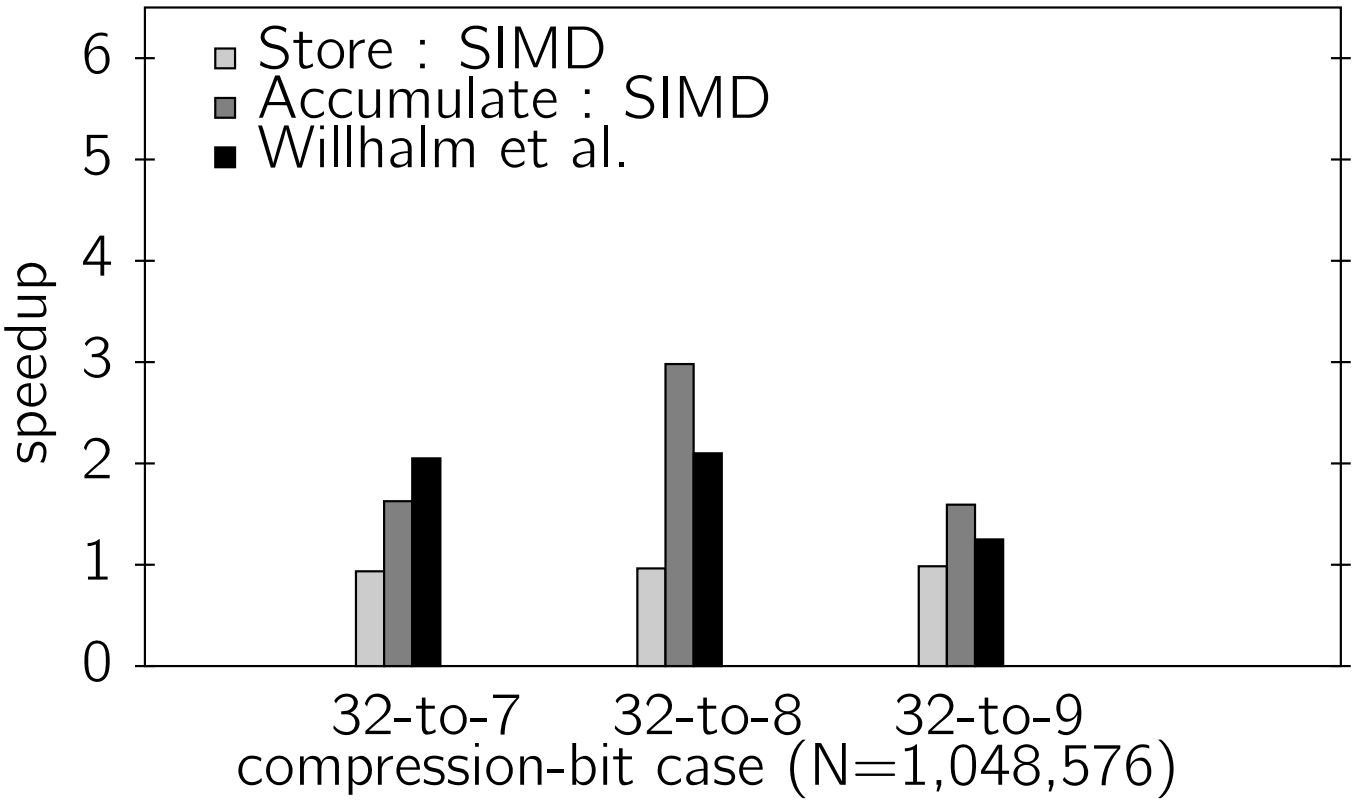
- Speedup versus optimized scalar implementation.

Speedup Comparison



- Why is our speedup better?
  - Serial version less optimized
  - In the *accumulate* case, we also do the addition in parallel
  - SIMD performance may vary on different hardware

Speedup Comparison: Core 2 Duo (Penryn)



- Model name : Intel(R) Core(TM)2 Duo CPU P9600

Advanced Vector Extensions (AVX)

- SSE5 was planned by AMD but canceled in favor of AVX (3DNow! Disaster 1998)
- SIMD-Register 128 bit $\rightarrow$ 256 bit
- CPUs with AVX
  - Intel : Sandy Bridge, Q1 2011
  - AMD : Bulldozer, Q3 2011, (first Server-Processor $\rightarrow$ Interlagos with AVX)
- AMD Bulldozer: Fused multiplyadd: A fused multiplyadd is a floating-point multiplyadd operation performed in one step
- A fast FMA can speed up and improve the accuracy of many computations that involve the accumulation of products
- Intel : FMA planned for Haswell processors in 2013