Tool zoo

Choosing the right vectorization method

Nicolai Behmann

December 5, 2015

Institute of Microelectronic Systems, Leibniz Universität Hannover

Table of contents

1. Motion estimation

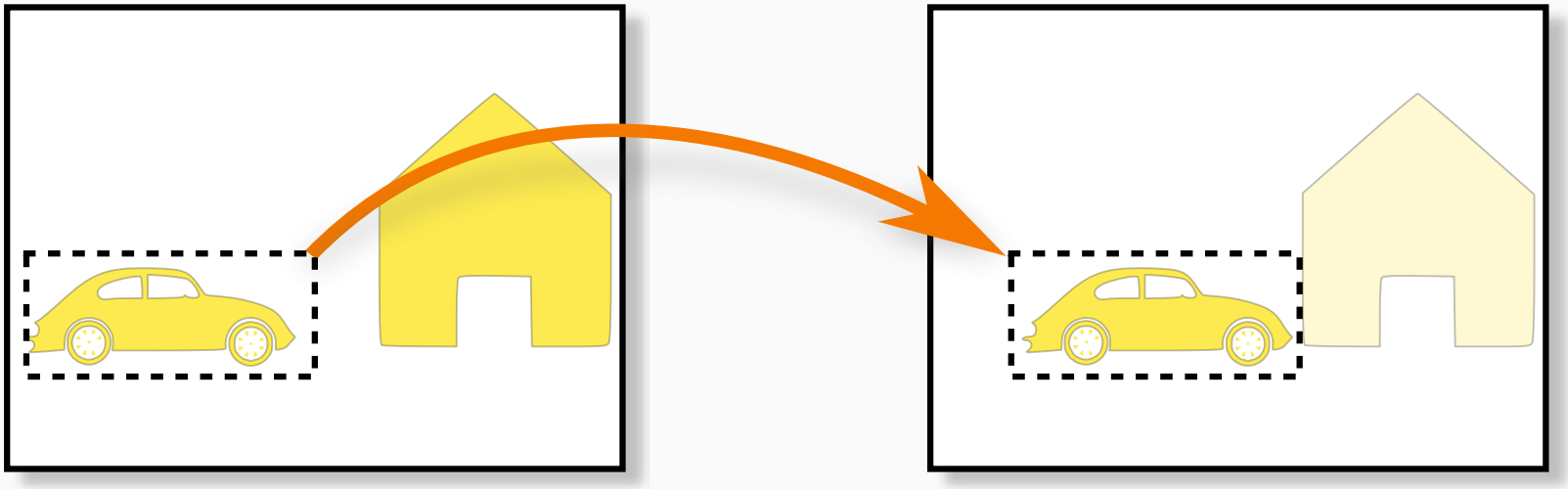
2. Vectorization methods

3. Performance evaluation

4. Conclusion

Motion estimation

Block-based motion estimation



Block-based motion estimation

Compare blocks by sum of pixelwise absolute difference.

```
for (i = 0; i < nBlocks_y; ++i)
      for (j = 0; j < nBlocks_x; ++j)
          for (n = 0; n < nCandidates; ++n)
              sad = calcSAD(...);
              // ... find max similarity, e.g. min sad
```

Vectorization methods

Autovectorization

```
for (sad=0, y=0; y < blockHeight; y++) {
    for (x=0; x < blockWidth; x++) {
        idx = y*linewidth + x;
        sad += abs(a[idx] - b[idx]);
}   }
```

Autovectorization

```
for (sad=0, y=0; y < blockHeight; y++) {
    for (x=0; x < blockWidth; x++) {
        idx = y*linewidth + x;
        sad += abs(a[idx] - b[idx]);
}   }
```

- Included in most C++ compilers
- Activation by optimization switch -O3

Autovectorization

```
for (sad=0, y=0; y < blockHeight; y++) {
    for (x=0; x < blockWidth; x++) {
        idx = y*linewidth + x;
        sad += abs(a[idx] - b[idx]);
}   }
```

- Included in most C++ compilers
- Activation by optimization switch `-O3`
- Improve performance by
  1. remove explicit index calculation
  2. using compile-time known loop length
  3. mark pointers with `restrict`
  4. align pointers

Autovectorization

```
for (sad=0, y=0; y < blockHeight; y++) {
    for (x=0; x < blockWidth; x++) {
        idx = y*linewidth + x;
        sad += abs(a[idx] - b[idx]);
}   }
```

- Included in most C++ compilers
- Activation by optimization switch `-O3`
- Improve performance by
  1. remove explicit index calculation
  2. using compile-time known loop length
  3. mark pointers with `restrict`
  4. align pointers
- Zero code adaption

OpenMP SIMD

```
for (sad=0, y=0, pA=a, pB=b; y < 16; ++y) {
    #pragma omp simd reduction(+:sad)
        aligned(pA,pB:16) linear(pA,pB:1)
    for (int x = 0; x < 16; x++)
        sad += std::abs(pA[x] - pB[x]);
    pA += lineWidth; pB += lineWidth;
}
```

OpenMP SIMD

```
for (sad=0, y=0, pA=a, pB=b; y < 16; ++y) {
    #pragma omp simd reduction(+:sad)
        aligned(pA,pB:16) linear(pA,pB:1)
    for (int x = 0; x < 16; x++)
        sad += std::abs(pA[x] - pB[x]);
    pA += lineWidth; pB += lineWidth;
}
```

- Introduced with OpenMP 4.0
- Possibility for many compiler hints

OpenMP SIMD

```
for (sad=0, y=0, pA=a, pB=b; y < 16; ++y) {
    #pragma omp simd reduction(+:sad)
        aligned(pA,pB:16) linear(pA,pB:1)
    for (int x = 0; x < 16; x++)
        sad += std::abs(pA[x] - pB[x]);
    pA += lineWidth; pB += lineWidth;
}
```

- Introduced with OpenMP 4.0
- Possibility for many compiler hints
- Very few code adaption

Array slicing notation

```
for (sad=0, y=0, pA=a, pB=b; y < 16; ++y) {
    tmp[0:16] = std::abs(pA[0:16] - pB[0:16]);
    sad += __sec_reduce_add(tmp[0:16]);
    pA += lineWidth; pB += lineWidth;
}
```

Array slicing notation

```
for (sad=0, y=0, pA=a, pB=b; y < 16; ++y) {
    tmp[0:16] = std::abs(pA[0:16] - pB[0:16]);
    sad += __sec_reduce_add(tmp[0:16]);
    pA += lineWidth; pB += lineWidth;
}
```

- Introduced with Cilk Plus
- Integrated in gcc 4.9+, few platforms

Array slicing notation

```
for (sad=0, y=0, pA=a, pB=b; y < 16; ++y) {
    tmp[0:16] = std::abs(pA[0:16] - pB[0:16]);
    sad += __sec_reduce_add(tmp[0:16]);
    pA += lineWidth; pB += lineWidth;
}
```

- Introduced with Cilk Plus
- Integrated in gcc 4.9+, few platforms
- Medium code adaption

Instruction-set intrinsics

```
__m128i rSad = _mm_setzero_si128();
for (y=0, pA=a, pB=b; y < 16; y++) {
    __m128i rA = _mm_load_si128(pA);
    __m128i rB = _mm_load_si128(pB);
    __m128i rT = _mm_sad_epu8(rA, rB);
    rSad += _mm_adds_epu16(rSad, rT);
    pA += lineWidth; pB += lineWidth;
}
sad = _mm_extract_epi16(rSad, 0)+_mm_extract_epi16(rSad, 4);
```

Instruction-set intrinsics

```
__m128i rSad = _mm_setzero_si128();
for (y=0, pA=a, pB=b; y < 16; y++) {
    __m128i rA = _mm_load_si128(pA);
    __m128i rB = _mm_load_si128(pB);
    __m128i rT = _mm_sad_epu8(rA, rB);
    rSad += _mm_adds_epu16(rSad, rT);
    pA += lineWidth; pB += lineWidth;
}
sad = _mm_extract_epi16(rSad, 0)+_mm_extract_epi16(rSad, 4);
```

- Low level accelerator access
- Few platforms, less portability

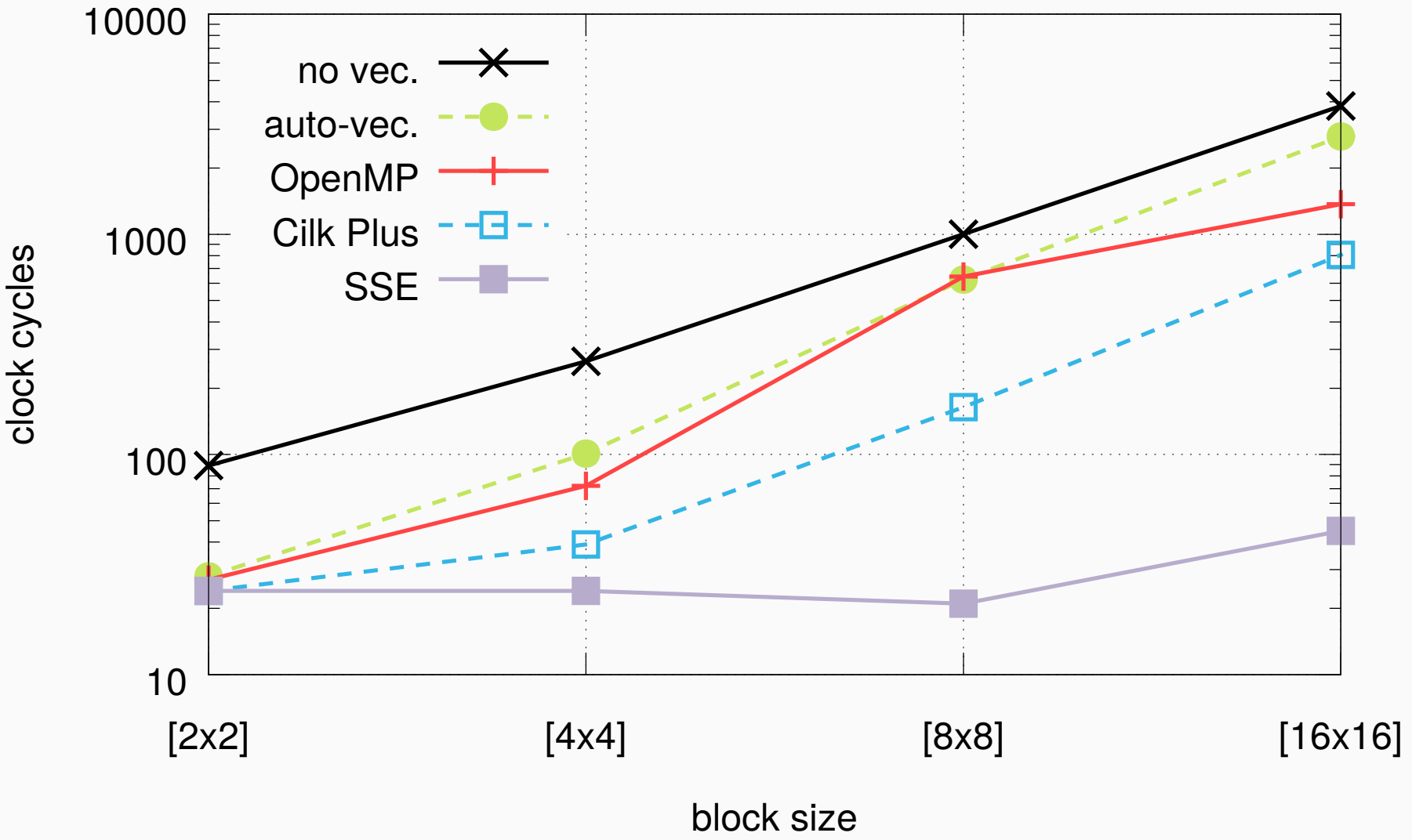
Instruction-set intrinsics

```
__m128i rSad = _mm_setzero_si128();
for (y=0, pA=a, pB=b; y < 16; y++) {
    __m128i rA = _mm_load_si128(pA);
    __m128i rB = _mm_load_si128(pB);
    __m128i rT = _mm_sad_epu8(rA, rB);
    rSad += _mm_adds_epu16(rSad, rT);
    pA += lineWidth; pB += lineWidth;
}
sad = _mm_extract_epi16(rSad, 0)+_mm_extract_epi16(rSad, 4);
```

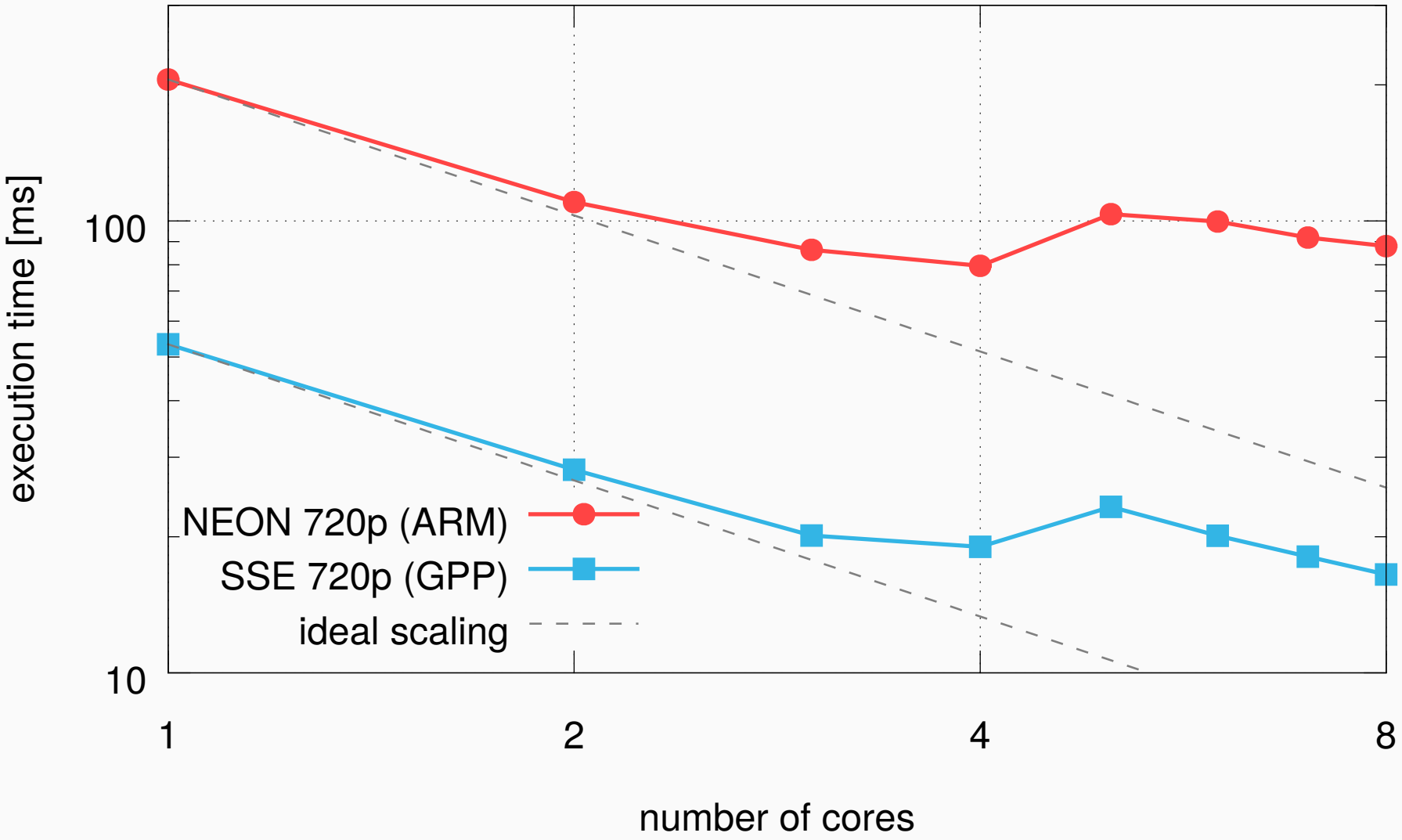
- Low level accelerator access
- Few platforms, less portability
- Huge code adaption

Performance evaluation

Cycle count comparison

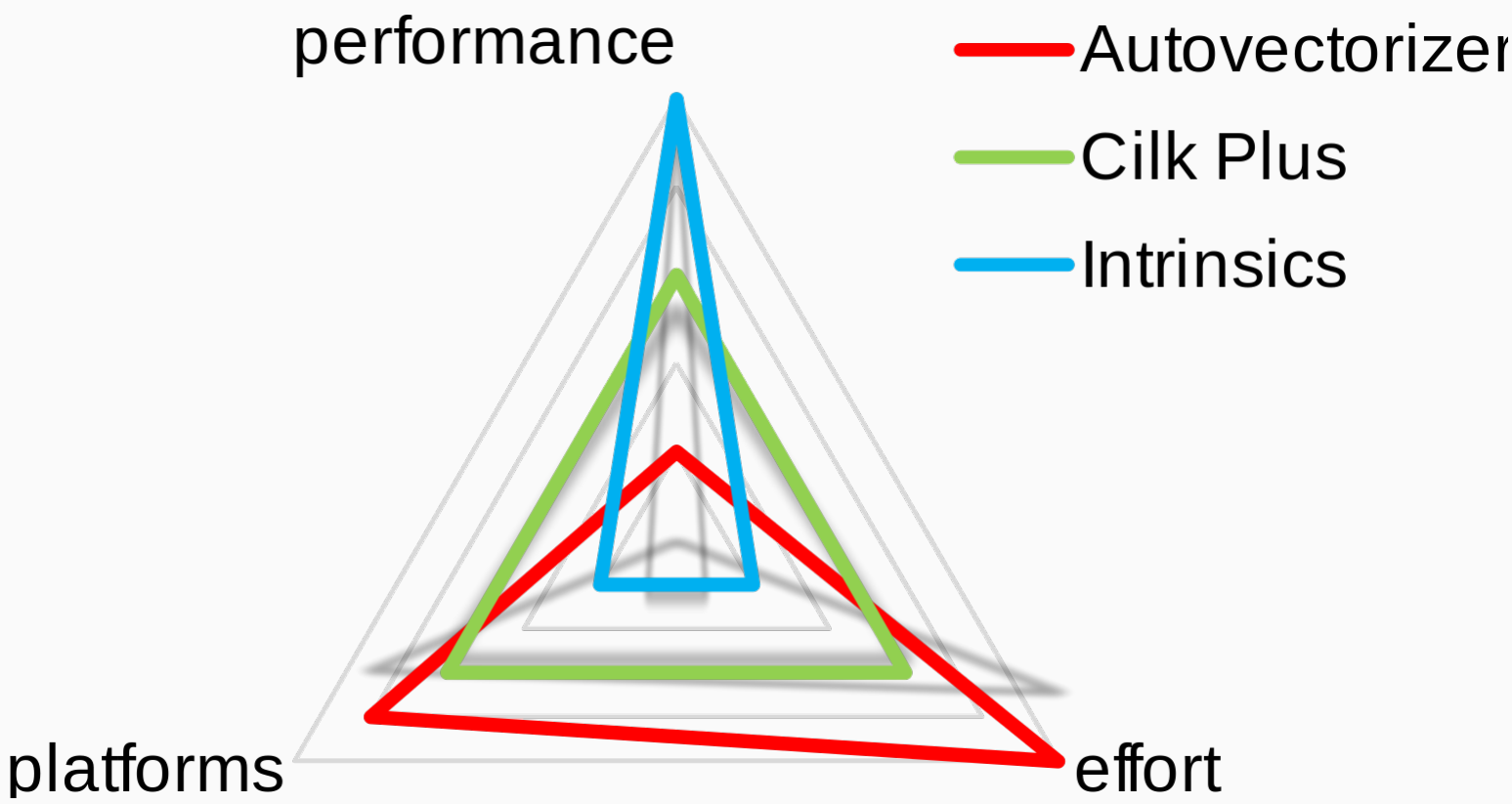


Additional outer parallelization

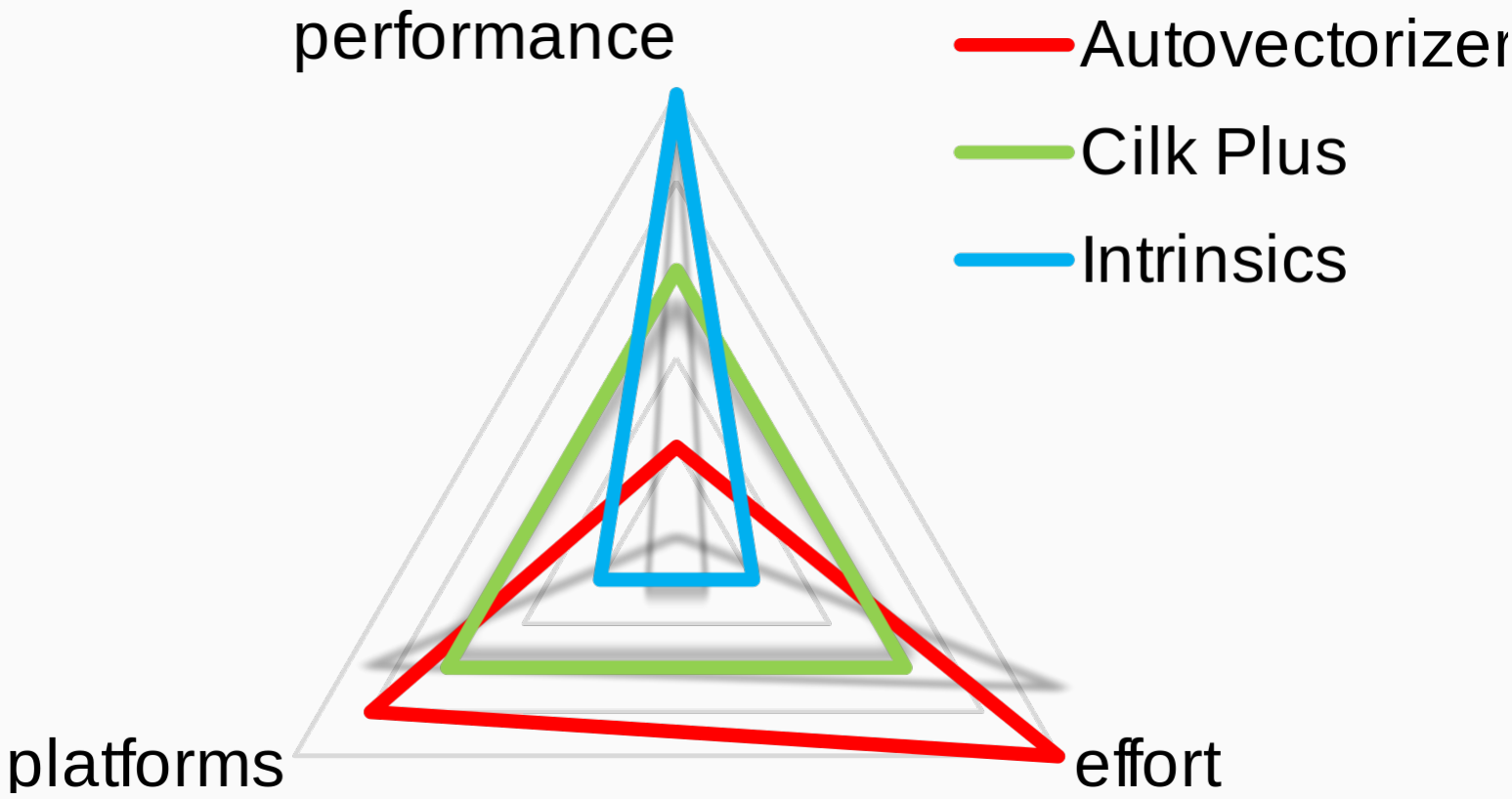


Conclusion

Summary


performance
Autovectorizer
Cilk Plus
Intrinsics
platforms
effort

Summary



Choice of vectorization method is strongly application specific.

`Boost.SIMD`, `Halide` and many others are also worth a try.

Questions?