MIPP: a Portable C++ SIMD Wrapper and its use for Error Correction Coding in 5G Standard

Adrien Cassagne[1,2] Olivier Aumage[1] Denis Barthou[1]
Camille Leroux[2] Christophe Jégo[2]

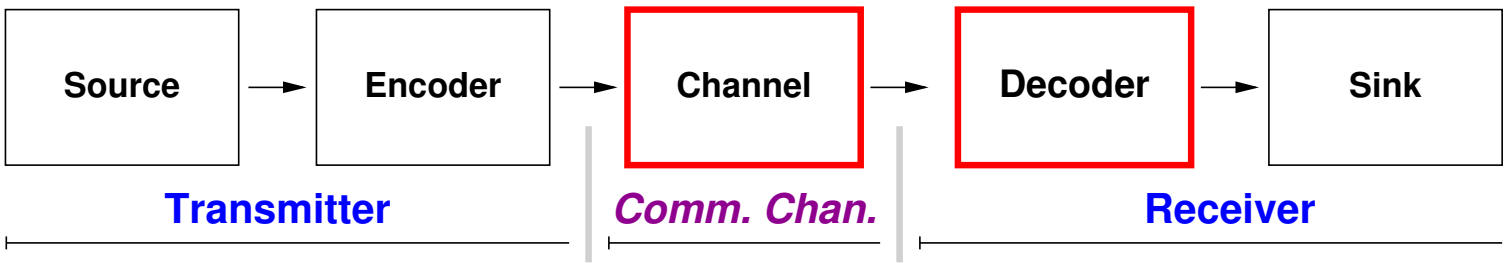
[1]Inria, Bordeaux Institute of Technology, U. of Bordeaux, LaBRI/CNRS, Bordeaux, France

[2]IMS/CNRS, Bordeaux, France

WPMVP, February 2018

Context: Error Correction Codes (ECC)

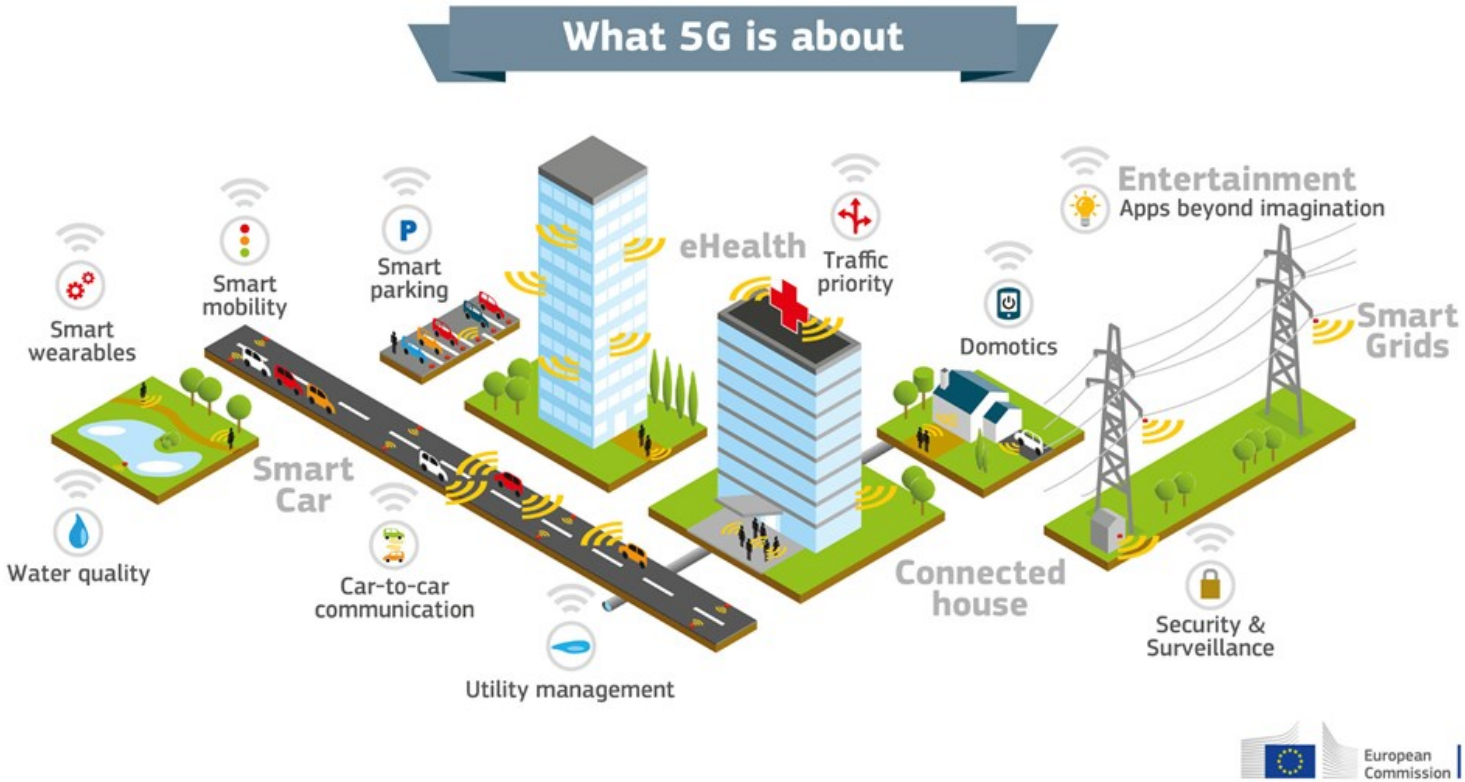
- Algorithms that enable **reliable delivery of digital data**
  - **Redundancy** for error correction
- Usually, **hardware implementation**
- Growing interest for **software implementation**
  - End-user utilization (low power consumption processors)
  - Algorithm validation (Monte-Carlo simulations)
  - **Requires performance**



The communication chain

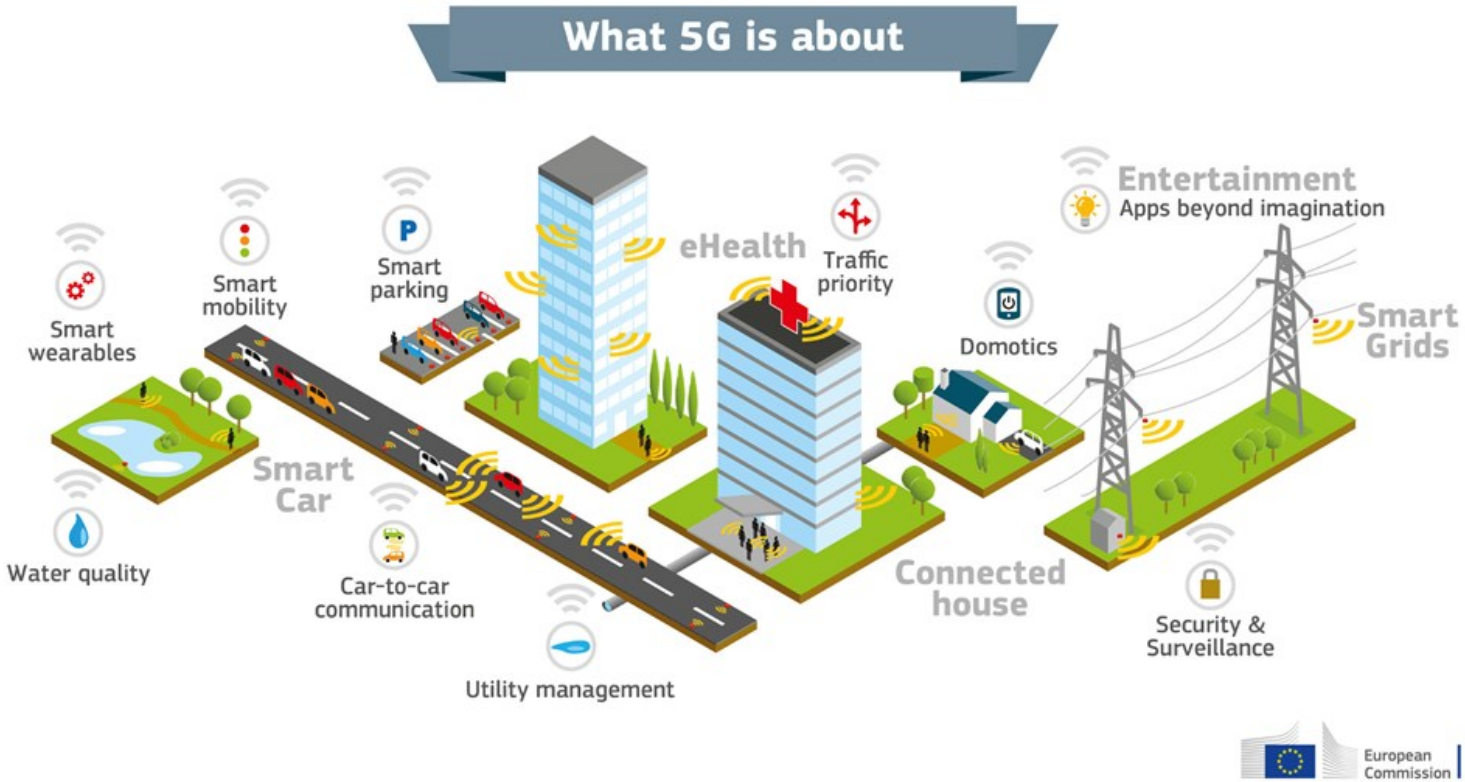
Context: 5G Standard

- **Many devices** connected to Internet via **wireless interfaces**
- Large zoo of devices: **Flexibility**
- Varied, complex ECC algorithms: **Expensive testing and validation**



Context: 5G Standard

- **Many devices** connected to Internet via **wireless interfaces**
- Large zoo of devices: **Flexibility**
- Varied, complex ECC algorithms: **Expensive testing and validation**
- $\Rightarrow$ **Portable SIMD library**



Proposal: MyIntrinsics++ (MIPP)

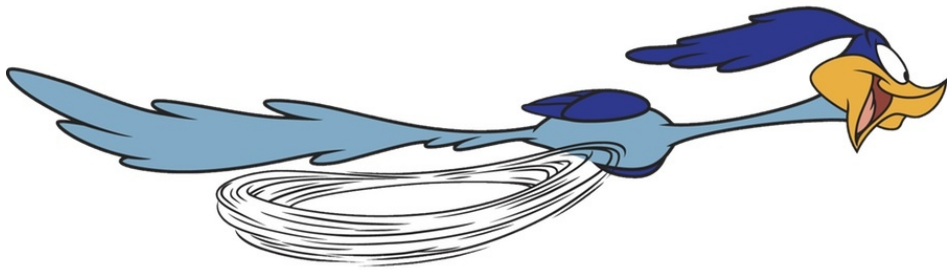
- SIMD C++ **wrapper**
  - Maximizes code portability
  - **Improves expressiveness** over intrinsics
  - **No dependencies** to other libraries

Proposal: MyIntrinsics++ (MIPP)

- SIMD C++ **wrapper**
  - Maximizes code portability
  - **Improves expressiveness** over intrinsics
  - **No dependencies** to other libraries
- Programming model **close to intrinsics**
  - Whenever possible, **one statement = one intrinsic**
  - One variable = one register allocation

Proposal: MyIntrinsics++ (MIPP)

- SIMD C++ **wrapper**
  - Maximizes code portability
  - **Improves expressiveness** over intrinsics
  - **No dependencies** to other libraries
- Programming model **close to intrinsics**
  - Whenever possible, **one statement = one intrinsic**
  - One variable = one register allocation



MIPP MIPP!

MIPP: Programming Interfaces

- The Low-Level MIPP Interface (low)
  - Similar to SIMD intrinsics: Untyped registers, typed operations
  - Portability through **template specialization** and **function inlining**

MIPP: Programming Interfaces

- The Low-Level MIPP Interface (low)
  - Similar to SIMD intrinsics: Untyped registers, typed operations
  - Portability through **template specialization** and **function inlining**
- The Medium-Level MIPP Interface (med.)
  - Typed registers, polymorphic operations
  - Based on MIPP Low Level Interface
  - Typing through **object encapsulation** and **operator overloading**

MIPP: Low Level Interface (low)

Similar to SIMD intrinsics: Untyped registers, typed operations

- Abstract SIMD **data register** (`mipp::reg`)
  - **The number of elements** in a register: `mipp::N<T>()`
- Abstract SIMD **mask register** (`mipp::msk`)
- Intrinsic functions wrapped into MIPP functions

```
1 template<> mipp::reg mipp::add<float>(mipp::reg a, mipp::reg b)
2 {
3   return (mipp::reg) _mm256_add_ps((__mm256) a, (__mm256) b);
4 }
5 template<> mipp::reg mipp::add<double>(mipp::reg a, mipp::reg b)
6 {
7   return (mipp::reg) _mm256_add_pd((__mm256d) a, (__mm256d) b);
8 }
```

  - **C++ template specialization technique**
- **Instruction set selection** based on preprocessor definitions

MIPP: Low Level Interface (low)

Similar to SIMD intrinsics: Untyped registers, typed operations

- Abstract SIMD **data register** (`mipp::reg`)
  - **The number of elements** in a register: `mipp::N<T>()`
- Abstract SIMD **mask register** (`mipp::msk`)
- Intrinsic functions wrapped into MIPP functions

```
1 template<> mipp::reg mipp::add<float>(mipp::reg a, mipp::reg b)
2 {
3   return (mipp::reg) _mm256_add_ps((__mm256) a, (__mm256) b);
4 }
5 template<> mipp::reg mipp::add<double>(mipp::reg a, mipp::reg b)
6 {
7   return (mipp::reg) _mm256_add_pd((__mm256d) a, (__mm256d) b);
8 }
```

  - **C++ template specialization technique**
- **Instruction set selection** based on preprocessor definitions

MIPP: Medium Level Interface (med.)

Typed registers, polymorphic operations

- Typed, portable SIMD **data register** (`mipp::Reg<T>`)
  - Contains a MIPP low data register (`mipp::reg`)
- Typed, portable SIMD **mask register** (`mipp::Msk<N>`)
  - Contains a MIPP low mask register (`mipp::msk`)
- Supports **operator overloading**
  - `mipp::Reg<float> a = 1.f, b = 2.f; auto c = a + b;`
- Eases the **register initializations** (broadcasts, sets and loads)
  - `mipp::Reg<int> a = 1, b = {1, 2, 3, 4}, c = ptr;`
- Defines an **aligned memory allocator** for the `std::vector<T,A>` class (`mipp::vector<T>`)

MIPP: Hello World!

```cpp
template <typename T>
void vecAdd(const std::vector<T> &A,
            const std::vector<T> &B,
                  std::vector<T> &C)
{
  // N elements per SIMD register
  constexpr int N = mipp::N<T>();

  // sizes verifications
  assert(A.size() == B.size());
  assert(A.size() == C.size());
  assert((A.size() % N) == 0);

  for(auto i = 0; i < A.size(); i += N)
  {
    mipp::Reg<T> rA = &A[i]; // SIMD load
    mipp::Reg<T> rB = &B[i]; // SIMD load

    mipp::Reg<T> rC = rA + rB; // SIMD addition

    rC.store(&C[i]); // SIMD store
  }
}
```

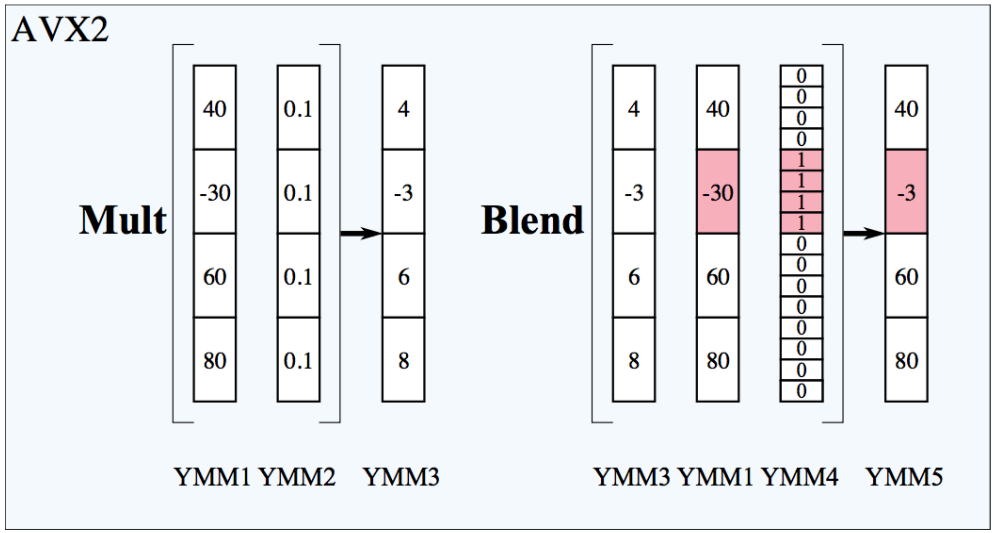
MIPP: Hello World!

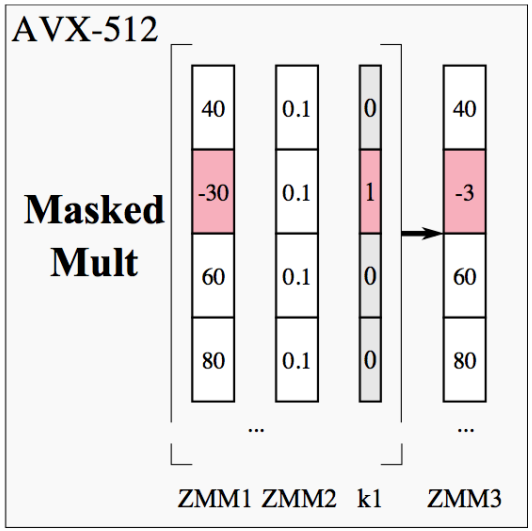
```cpp
template <typename T>
void vecAdd(const std::vector<T> &A,
            const std::vector<T> &B,
                  std::vector<T> &C)
{
  // N elements per SIMD register
  constexpr int N = mipp::N<T>();

  // sizes verifications
  assert(A.size() == B.size());
  assert(A.size() == C.size());
  assert((A.size() % N) == 0);

  for(auto i = 0; i < A.size(); i += N)
  {
    mipp::Reg<T> rA = &A[i]; // SIMD load
    mipp::Reg<T> rB = &B[i]; // SIMD load

    mipp::Reg<T> rC = rA + rB; // SIMD addition

    rC.store(&C[i]); // SIMD store
  }
}
```

- MIPP operates at **register level**
- Same code can **work on various data types** and instruction sets

MIPP: Masking Support

- In the AVX-512 instruction set:
  - Height **dedicated hardware mask registers** (k0, k1, ..., k7)
  - Some **operations can be masked**

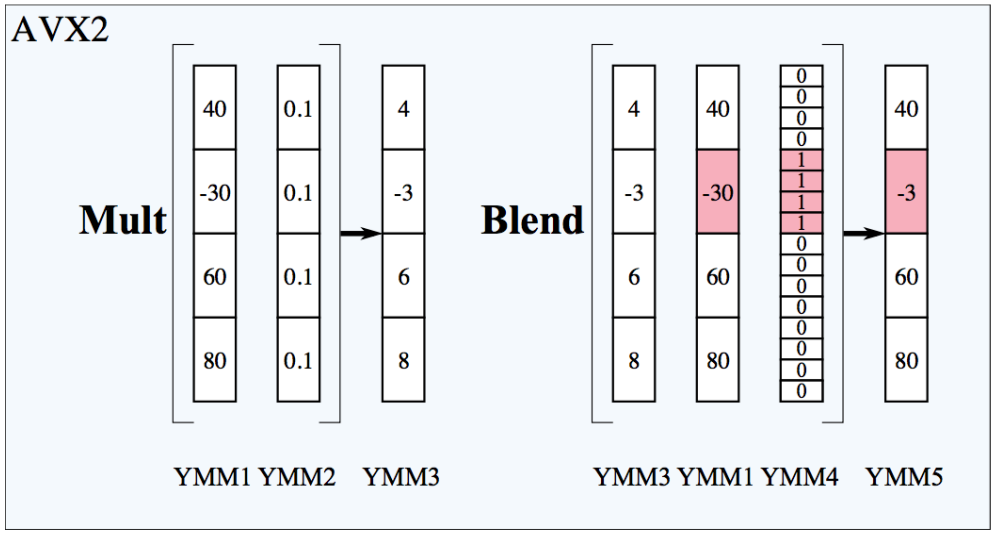


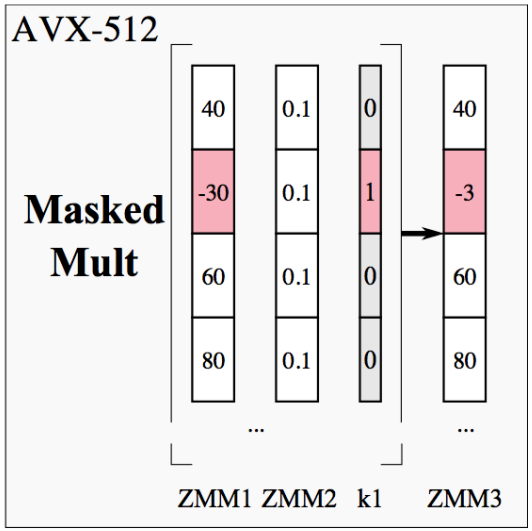


Picture from colfaxresearch.com

MIPP: Masking Support

- In the AVX-512 instruction set:
  - Height **dedicated hardware mask registers** (`k0, k1, ..., k7`)
  - Some **operations can be masked**

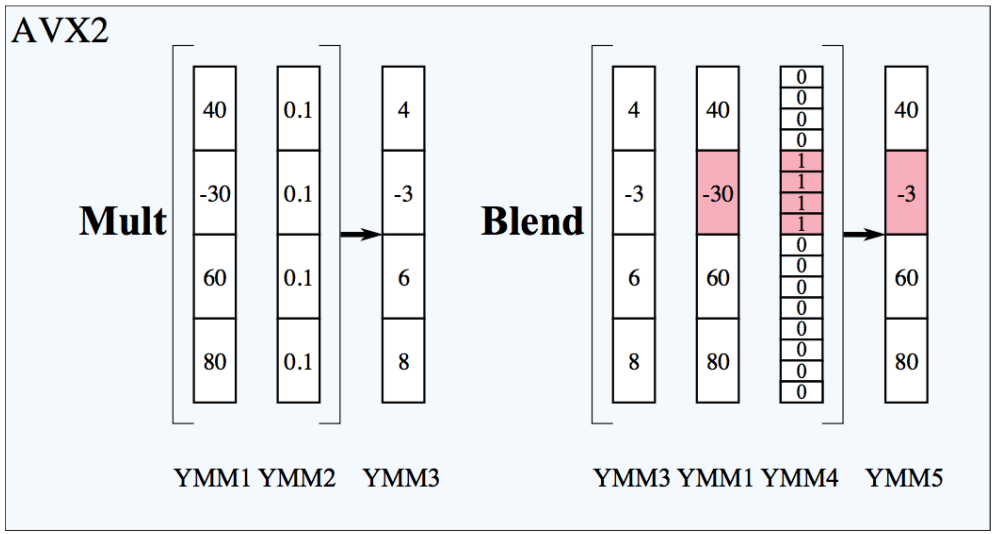


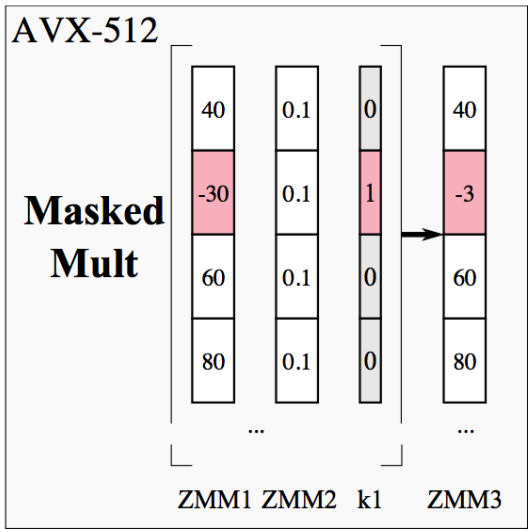


Picture from colfaxresearch.com

- MIPP takes advantage of the masked operations with a dedicated function: `mipp::mask<T,mipp::op>(m, src, r1, r2)`

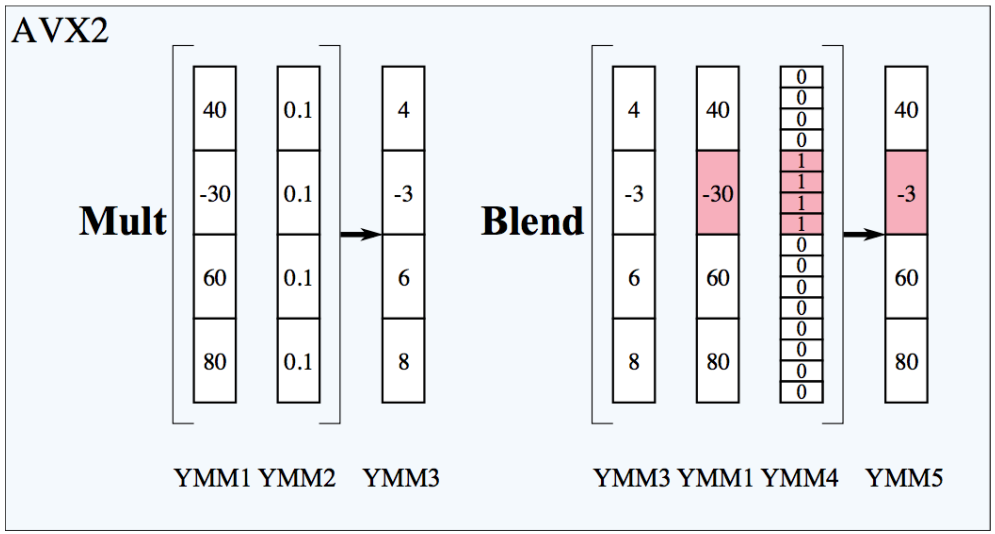
MIPP: Masking Support (example)

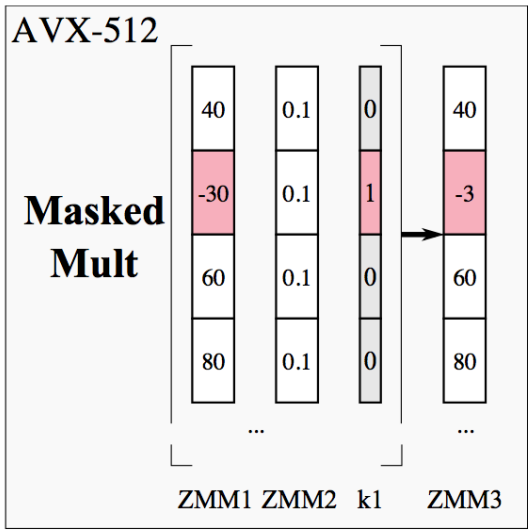





```
1 mipp::Reg<         float    > ZMM1 = {   40,  -30,    60,    80};
2 mipp::Reg<         float    > ZMM2 = 0.1; // broadcast
3 mipp::Msk<mipp::N<float>()> k1   = {false, true, false, false};
4
5 // ZMM3 = k1 ? ZMM1 * ZMM2 : ZMM1;
6 auto ZMM3 = mipp::mask<float, mipp::mul>(k1, ZMM1, ZMM1, ZMM2);
7
8 std::cout << ZMM3 << std::endl; // [   40,   -3,    60,    80]
```

MIPP: Masking Support (example)






```
1 mipp::Reg<          float    > ZMM1 = {   40,  -30,    60,    80};
2 mipp::Reg<          float    > ZMM2 = 0.1; // broadcast
3 mipp::Msk<mipp::N<float>()> k1   = {false, true, false, false};
4
5 // ZMM3 = k1 ? ZMM1 * ZMM2 : ZMM1;
6 auto ZMM3 = mipp::mask<float, mipp::mul>(k1, ZMM1, ZMM1, ZMM2);
7
8 std::cout << ZMM3 << std::endl; // [   40,   -3,    60,    80]
```

Experimentation Protocol

- Experiments
  - **5G standard** related case studies
  - Comparison to **related works** on Mandelbrot set
- Tests are **mono-threaded**
- GNU C++ compiler version 5

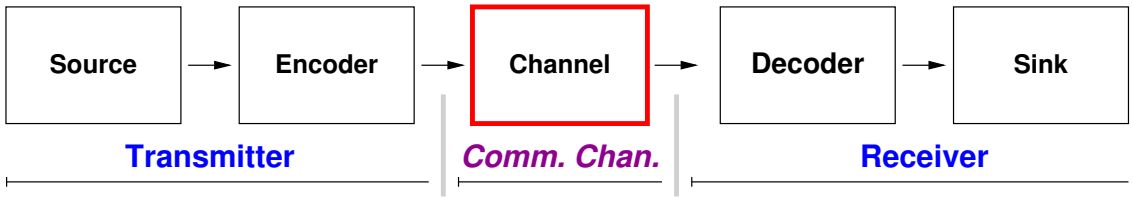
Instr. set	NEON	SSE / AVX	AVX-512
Name	**Exynos5422**	**Core i5-5300U**	**Xeon Phi 7230**
Year	2014	2015	2016
Vendor	Samsung	Intel	Intel
Arch.	ARMv7 Cortex A15	x86-64 Broadwell	Knights Landing
Cores/Freq.	4/2.0 GHz	2/2.3 GHz	64/1.3 GHz
LLC	2 MB L2	3 MB L3	32 MB L2
TDP	∼4 W	15 W	215 W

Experimentation Protocol

- Experiments
  - **5G standard** related case studies
  - Comparison to **related works** on Mandelbrot set
- Tests are **mono-threaded**
- GNU C++ compiler version 5

Instr. set	NEON	SSE / AVX	AVX-512
Name	**Exynos5422**	**Core i5-5300U**	**Xeon Phi 7230**
Year	2014	2015	2016
Vendor	Samsung	Intel	Intel
Arch.	ARMv7 Cortex A15	x86-64 Broadwell	Knights Landing
Cores/Freq.	4/2.0 GHz	2/2.3 GHz	64/1.3 GHz
LLC	2 MB L2	3 MB L3	32 MB L2
TDP	∼4 W	15 W	215 W

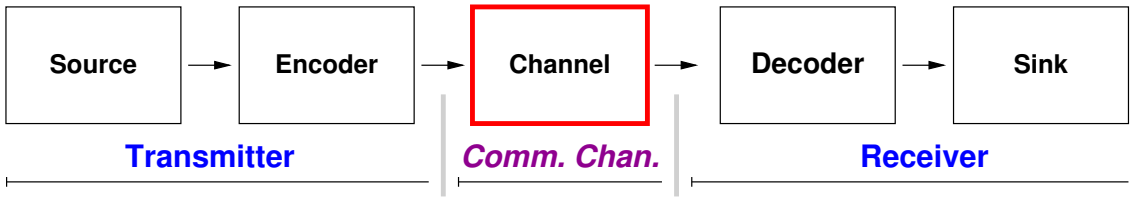
5G Case Studies: Box-Muller Transform




```
1  void BoxMuller(const std::vector<float> &U
2                       std::vector<float> &Z)
3  {
4    constexpr auto N = mipp::N<float>();
5    const auto nElmts = U.size();
6    const auto twoPi = 2.f * 3.141592f;
7    for (auto i = 0; i < nElmts; i += N * 2)
8    {
9      auto u1 = mipp::Reg<float>(&U[    i]);
10     auto u2 = mipp::Reg<float>(&U[N +i]);
11     auto r = mipp::sqrt(mipp::log(u1)*-2.f);
12     auto theta  = u2 * twoPi;
13     mipp::Reg<float> sinTheta, cosTheta;
14     mipp::sincos(theta, sinTheta, cosTheta);
15     auto z1 = r * cosTheta;
16     auto z2 = r * sinTheta;
17     z1.store(&Z[    i]);
18     z2.store(&Z[N +i]);
19   }
20 }
```

- **20% – 50% of simulation time**
- MIPP includes `sqrt`, `log`, `exp`, `sin`, `cos` and `sincos` math functions

5G Case Studies: Box-Muller Transform

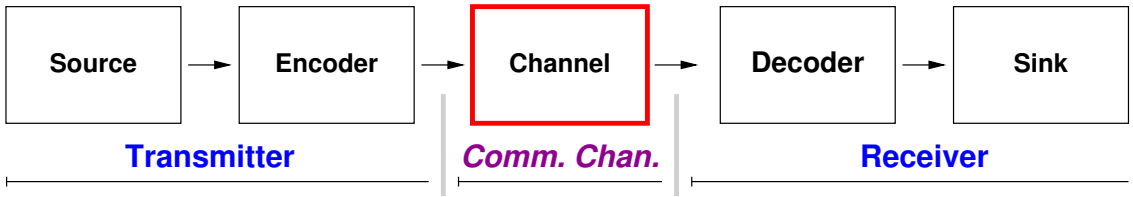



```
1  void BoxMuller(const std::vector<float> &U
2                 std::vector<float> &Z)
3  {
4    constexpr auto N = mipp::N<float>();
5    const auto nElmts = U.size();
6    const auto twoPi = 2.f * 3.141592f;
7    for (auto i = 0; i < nElmts; i += N * 2)
8    {
9      auto u1 = mipp::Reg<float>(&U[   i]);
10     auto u2 = mipp::Reg<float>(&U[N +i]);
11     auto r = mipp::sqrt(mipp::log(u1)*-2.f);
12     auto theta  = u2 * twoPi;
13     mipp::Reg<float> sinTheta, cosTheta;
14     mipp::sincos(theta, sinTheta, cosTheta);
15     auto z1 = r * cosTheta;
16     auto z2 = r * sinTheta;
17     z1.store(&Z[   i]);
18     z2.store(&Z[N +i]);
19   }
20 }
```

- **20% – 50% of simulation time**
- MIPP includes `sqrt`, `log`, `exp`, `sin`, `cos` and `sincos` math functions

	NEON	SSE	AVX	AVX-512
SIMD size	4	4	8	16
Seq. T/P (Mb/s)	13.2	46.7	42.5	6.6
MIPP T/P (Mb/s)	40.9	107.4	178.3	95.1
Speedup	×3.1	×2.3	×4.2	×14.4

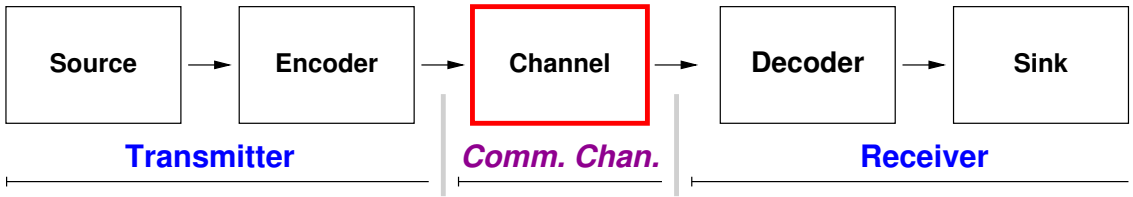
5G Case Studies: Quantizer




```
1  void Quantizer(const std::vector<float > &Y,
2                 std::vector<int8_t> &Ysv,
3                 const unsigned s, const unsigned v) {
4    constexpr auto N = mipp::N<float >();
5    const auto pow2 = mipp::Reg<float >(1 << v);
6    const float qMax = (1 << (s-2)) + (1 << (s-2)) -1;
7    const float qMin = -qMax;
8    for (auto y = 0; y < Y.size(); y += 4 * N) {
9      // implicit loads and q = 2^v * y +/- 0.5
10     auto q32_0 = mipp::round(pow2 * &Y[y + 0*N]);
11     auto q32_1 = mipp::round(pow2 * &Y[y + 1*N]);
12     auto q32_2 = mipp::round(pow2 * &Y[y + 2*N]);
13     auto q32_3 = mipp::round(pow2 * &Y[y + 3*N]);
14     // convert float to int32_t
15     auto q32i_0 = mipp::cvt<int32_t>(q32_0);
16     auto q32i_1 = mipp::cvt<int32_t>(q32_1);
17     auto q32i_2 = mipp::cvt<int32_t>(q32_2);
18     auto q32i_3 = mipp::cvt<int32_t>(q32_3);
19     // pack four int32_t in two int16_t
20     auto q16i_0 = mipp::pack<int32_t,int16_t>(q32i_0,
21                                               q32i_1);
22     auto q16i_1 = mipp::pack<int32_t,int16_t>(q32i_2,
23                                               q32i_3);
24     // pack two int16_t in one int8_t
25     auto q8i = mipp::pack<int16_t,int8_t>(q16i_0, q16i_1);
26     // saturation (psi function)
27     auto q8is = mipp::sat(q8i, qMin, qMax);
28     q8is.store(&Ysv[y]);
29   }
30 }
```

- **Converts** 32-bit floating-point numbers into 8-bit integers
- **Auto-vectorization is very bad** when data conversions are involved
- `MIPP` provides `round`, `cvt` and `pack` functions

5G Case Studies: Quantizer

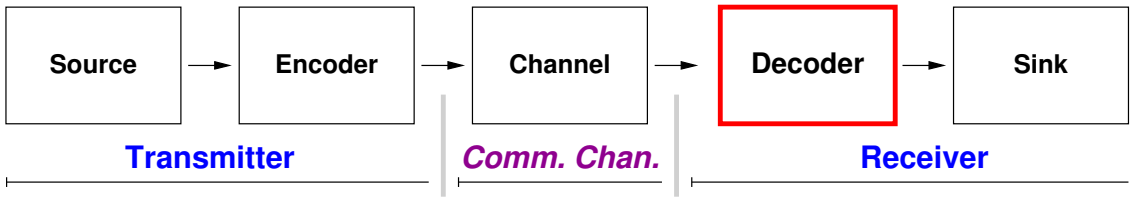



```
1  void Quantizer(const std::vector<float> &Y,
2                 std::vector<int8_t> &Ysv,
3                 const unsigned s, const unsigned v) {
4    constexpr auto N = mipp::N<float>();
5    const auto pow2 = mipp::Reg<float>(1 << v);
6    const float qMax = (1 << (s-2)) + (1 << (s-2)) -1;
7    const float qMin = -qMax;
8    for (auto y = 0; y < Y.size(); y += 4 * N) {
9      // implicit loads and q = 2^v * y +/- 0.5
10     auto q32_0 = mipp::round(pow2 * &Y[y + 0*N]);
11     auto q32_1 = mipp::round(pow2 * &Y[y + 1*N]);
12     auto q32_2 = mipp::round(pow2 * &Y[y + 2*N]);
13     auto q32_3 = mipp::round(pow2 * &Y[y + 3*N]);
14     // convert float to int32_t
15     auto q32i_0 = mipp::cvt<int32_t>(q32_0);
16     auto q32i_1 = mipp::cvt<int32_t>(q32_1);
17     auto q32i_2 = mipp::cvt<int32_t>(q32_2);
18     auto q32i_3 = mipp::cvt<int32_t>(q32_3);
19     // pack four int32_t in two int16_t
20     auto q16i_0 = mipp::pack<int32_t,int16_t>(q32i_0,
21                                               q32i_1);
22     auto q16i_1 = mipp::pack<int32_t,int16_t>(q32i_2,
23                                               q32i_3);
24     // pack two int16_t in one int8_t
25     auto q8i = mipp::pack<int16_t,int8_t>(q16i_0, q16i_1);
26     // saturation (psi function)
27     auto q8is = mipp::sat(q8i, qMin, qMax);
28     q8is.store(&Ysv[y]);
29   }
30 }
```

- **Converts** 32-bit floating-point numbers into 8-bit integers
- **Auto-vectorization is very bad** when data conversions are involved
- `MIPP` provides `round`, `cvt` and `pack` functions

	NEON	SSE	AVX
SIMD size	4-16	4-16	8-32
Seq. T/P (Mb/s)	65.3	227.0	218.2
MIPP T/P (Mb/s)	300.6	3541.4	5628.3
Speedup	×4.6	×15.6	×25.8

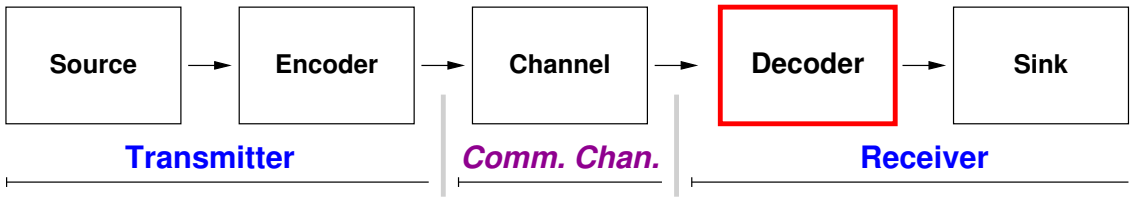
5G Case Studies: LDPC Codes Decoding (BP Min-Sum)




```
1  template <typename T>
2  void DecBP(const std::vector<std::vector<unsigned>> &H
3                  std::vector<mipp::Reg  <T       >> &VN
4                  std::vector<mipp::Reg  <T       >> &M,
5                  std::vector<mipp::Reg  <T       >> &C
6             const unsigned                           nIte) {
7    const auto max = std::numeric_limits<T>::max();
8    const auto zero = mipp::Reg<T>(0);
9    for (auto i = 0; i < nIte; i++) {
10     auto mRead = 0, mWrite = 0;
11     for (auto c = 0; c < H.size(); c++) {
12       constexpr auto N = mipp::N<T>();
13       auto sign = mipp::Msk<N>(false);
14       auto min1 = mipp::Reg<T>(max);
15       auto min2 = mipp::Reg<T>(max);
16       for (auto v = 0; v < H[c].size(); v++) {
17         C[v] = VN[H[c][v]] - M[mRead++];
18         auto cabs = mipp::abs(C[v]);
19         auto ctmp = min1;
20         sign ^= mipp::sign(C[v]);
21         min1  = mipp::min(min1,             cabs       );
22         min2  = mipp::min(min2, mipp::max(cabs, ctmp));
23       }
24       auto cst1 = mipp::blend(zero, min2, zero > min2);
25       auto cst2 = mipp::blend(zero, min1, zero > min1);
26       for (auto v = 0; v < H[c].size(); v++) {
27         auto cval = C[v];
28         auto cabs = mipp::abs(cval);
29         auto cres = mipp::blend(cst1, cst2, cabs == min1);
30         auto csig = sign ^ mipp::sign(cval);
31              cres = mipp::copysign(cres, csig);
32         M[mWrite++] = cres;
33         VN[H[c][v]] = C[v] + cres;
34 } } } }
```

- **Same code for various data types** (`double`, `float`, `int32_t`, `int16_t`)
- Decodes multiple frames (`N`) in parallel (SIMD **inter-frame strategy**)

5G Case Studies: LDPC Codes Decoding (BP Min-Sum)

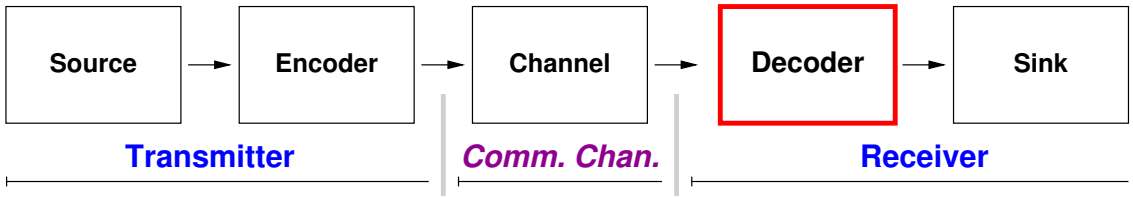



```
1  template <typename T>
2  void DecBP(const std::vector<std::vector<unsigned>> &H
3                   std::vector<mipp::Reg  <T       >> &VN
4                   std::vector<mipp::Reg  <T       >> &M,
5                   std::vector<mipp::Reg  <T       >> &C
6             const unsigned                            nIte) {
7    const auto max = std::numeric_limits<T>::max();
8    const auto zero = mipp::Reg<T>(0);
9    for (auto i = 0; i < nIte; i++) {
10     auto mRead = 0, mWrite = 0;
11     for (auto c = 0; c < H.size(); c++) {
12       constexpr auto N = mipp::N<T>();
13       auto sign = mipp::Msk<N>(false);
14       auto min1 = mipp::Reg<T>(max);
15       auto min2 = mipp::Reg<T>(max);
16       for (auto v = 0; v < H[c].size(); v++) {
17         C[v] = VN[H[c][v]] - M[mRead++];
18         auto cabs = mipp::abs(C[v]);
19         auto ctmp = min1;
20         sign ^= mipp::sign(C[v]);
21         min1  = mipp::min(min1,            cabs       );
22         min2  = mipp::min(min2, mipp::max(cabs, ctmp));
23       }
24       auto cst1 = mipp::blend(zero, min2, zero > min2);
25       auto cst2 = mipp::blend(zero, min1, zero > min1);
26       for (auto v = 0; v < H[c].size(); v++) {
27         auto cval = C[v];
28         auto cabs = mipp::abs(cval);
29         auto cres = mipp::blend(cst1, cst2, cabs == min1);
30         auto csig = sign ^ mipp::sign(cval);
31              cres = mipp::copysign(cres, csig);
32         M[mWrite++] = cres;
33         VN[H[c][v]] = C[v] + cres;
34 } } } }
```

- **Same code for various data types** (`double`, `float`, `int32_t`, `int16_t`)
- Decodes multiple frames (`N`) in parallel (SIMD **inter-frame strategy**)

	NEON	SSE	AVX
SIMD size	8	8	16
Seq. T/P (Mb/s)	0.9	3.4	3.5
MIPP T/P (Mb/s)	8.3	30.3	53.2
Speedup	×9.7	×8.8	×15.2

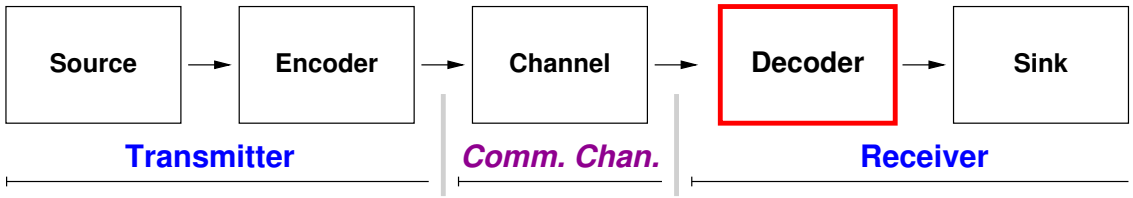
5G Case Studies: Polar Codes Decoding (SC)




```
1  template <typename T>
2  mipp::Reg<T> f_SIMD(const mipp::Reg<T> &la,
3                      const mipp::Reg<T> &lb)
4  {
5    auto abs_la  = mipp::abs(la);
6    auto abs_lb  = mipp::abs(lb);
7    auto min_abs = mipp::min(abs_la, abs_lb);
8    auto sign    = mipp::sign(la ^ lb);
9    // neg(Reg, Msk) = Msk ? -Reg : Reg;
10   return mipp::neg(min_abs, sign);
11 }
12
13 template <typename T, int N = mipp::N<T>()>
14 mipp::Reg<T> g_SIMD(const mipp::Reg<T> &la,
15                     const mipp::Reg<T> &lb,
16                     const mipp::Msk<N> &sa)
17 {
18   return mipp::neg(la, sa) + lb;
19 }
20
21 template <int N>
22 mipp::Msk<N> h_SIMD(const mipp::Msk<N>& sa,
23                     const mipp::Msk<N>& sb)
24 {
25   return sa ^ sb;
26 }
```

- **Same code for various data types** (`double`, `float`, `int16_t`, `int8_t`)
- Decodes frame per frame (SIMD **intra-frame strategy**)

5G Case Studies: Polar Codes Decoding (SC)




```
1  template <typename T>
2  mipp::Reg<T> f_SIMD(const mipp::Reg<T> &la,
3                      const mipp::Reg<T> &lb)
4  {
5    auto abs_la  = mipp::abs(la);
6    auto abs_lb  = mipp::abs(lb);
7    auto min_abs = mipp::min(abs_la, abs_lb);
8    auto sign    = mipp::sign(la ^ lb);
9    // neg(Reg, Msk) = Msk ? -Reg : Reg;
10   return mipp::neg(min_abs, sign);
11 }
12
13 template <typename T, int N = mipp::N<T>()>
14 mipp::Reg<T> g_SIMD(const mipp::Reg<T> &la,
15                     const mipp::Reg<T> &lb,
16                     const mipp::Msk<N> &sa)
17 {
18   return mipp::neg(la, sa) + lb;
19 }
20
21 template <int N>
22 mipp::Msk<N> h_SIMD(const mipp::Msk<N>& sa,
23                     const mipp::Msk<N>& sb)
24 {
25   return sa ^ sb;
26 }
```

- **Same code for various data types** (`double`, `float`, `int16_t`, `int8_t`)
- Decodes frame per frame (SIMD **intra-frame strategy**)

	NEON	SSE	AVX
SIMD size	16	16	32
Seq. T/P (Mb/s)	48.0	120.1	127.1
MIPP T/P (Mb/s)	148.7	528.3	483.0
Speedup	×3.1	×4.4	×3.8

Related works: SIMD Wrappers

Two main strategies:

1. **operator overloading** (used in `MIPP`)
2. **expression templates**, can automatically match instructions like Fused Multiply–Add (FMA, $d = a \times b + c$)

General Information		Instruction Set				Data Type						Features	
	Name	SSE 128-bit	AVX 256-bit	AVX-512 512-bit	NEON 128-bit	Float 64	Float 32	Integer 64	Integer 32	Integer 16	Integer 8	Math Func.	C++ Technique
Library	MIPP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	Op. overload.
	VCL	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	Op. overload.
	simdpp	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	Expr. templ.
	T-SIMD	✓	✓	✗	✓	✗	✓	✗	✓	✓	✓	✗	Op. overload.
	Vc	✓	✓	✗	✗	✓	✓	✓	✓	✓	✗	✓	Op. overload.
	xsimd	✓	✓	✗	✗	✓	✓	✓	✓	✗	✗	✓	Op. overload.
	Boost.SIMD	✓	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	Expr. templ.
	bSIMD	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	Expr. templ.

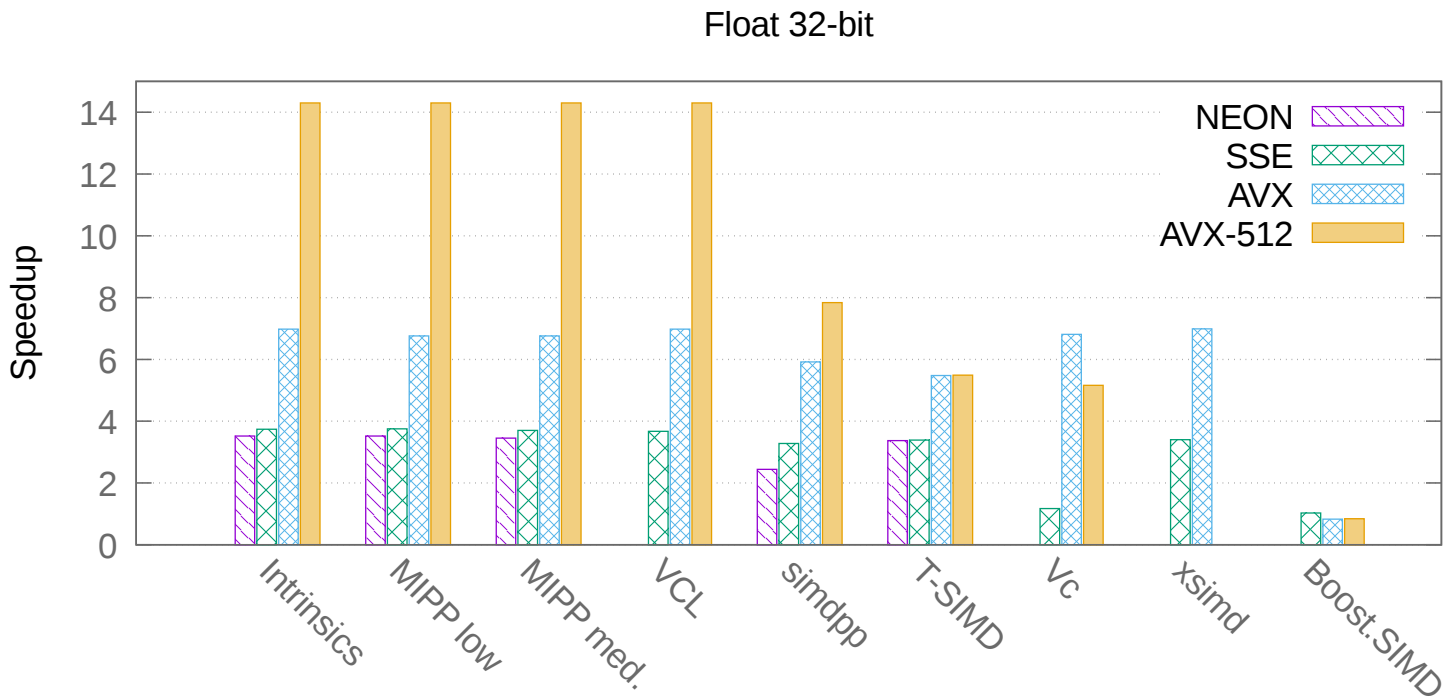
Related works: Comparison on the Mandelbrot Set

- **Computational intensive** kernel
- Need to manage an **early termination**
- C++ code is freely available[1]

[1] Full code is available online: https://gitlab.inria.fr/acassagn/mandelbrot

Related works: Comparison on the Mandelbrot Set

- **Computational intensive** kernel
- Need to manage an **early termination**
- C++ code is freely available[1]



[1] Full code is available online: https://gitlab.inria.fr/acassagn/mandelbrot

Conclusion and Future Works

Conclusion:

1. **SIMD wrapper** dedicated to digital communication algorithms

Conclusion and Future Works

Conclusion:

1. **SIMD wrapper** dedicated to digital communication algorithms
2. Matches both **simulation** and **embedded software** constraints

Conclusion and Future Works

Conclusion:

1. **SIMD wrapper** dedicated to digital communication algorithms
2. Matches both **simulation** and **embedded software** constraints
3. **Outperforms** most ECC software solutions on CPUs

Conclusion and Future Works

Conclusion:

1. **SIMD wrapper** dedicated to digital communication algorithms
2. Matches both **simulation** and **embedded software** constraints
3. **Outperforms** most ECC software solutions on CPUs
4. **Used by hardware designers** in academic and industrial research[2]

[2] AFF3CT Open-source toolbox: http://aff3ct.github.io

Conclusion and Future Works

Conclusion:

1. **SIMD wrapper** dedicated to digital communication algorithms
2. Matches both **simulation** and **embedded software** constraints
3. **Outperforms** most ECC software solutions on CPUs
4. **Used by hardware designers** in academic and industrial research[2]

Future works:

- Wraps the new **ARM Scalable Vector Extension** (SVE)
  - Predicate registers would likely map on `MIPP` masks
  - Main challenge will be to deal with **agnostic vector sizes**

[2] `AFF3CT` Open-source toolbox: `http://aff3ct.github.io`

Thank you!

- MIPP code: https://github.com/aff3ct/MIPP
- AFF3CT toolbox: https://github.com/aff3ct/aff3ct