FUJITSU

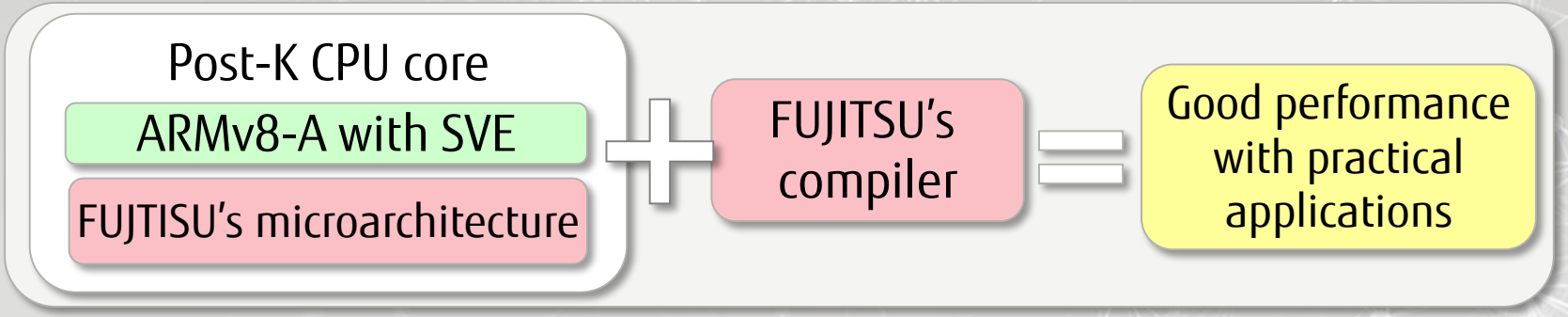
shaping tomorrow with you

ARMv8-A Scalable Vector Extension for Post-K

Post-K Supports New SIMD Extension

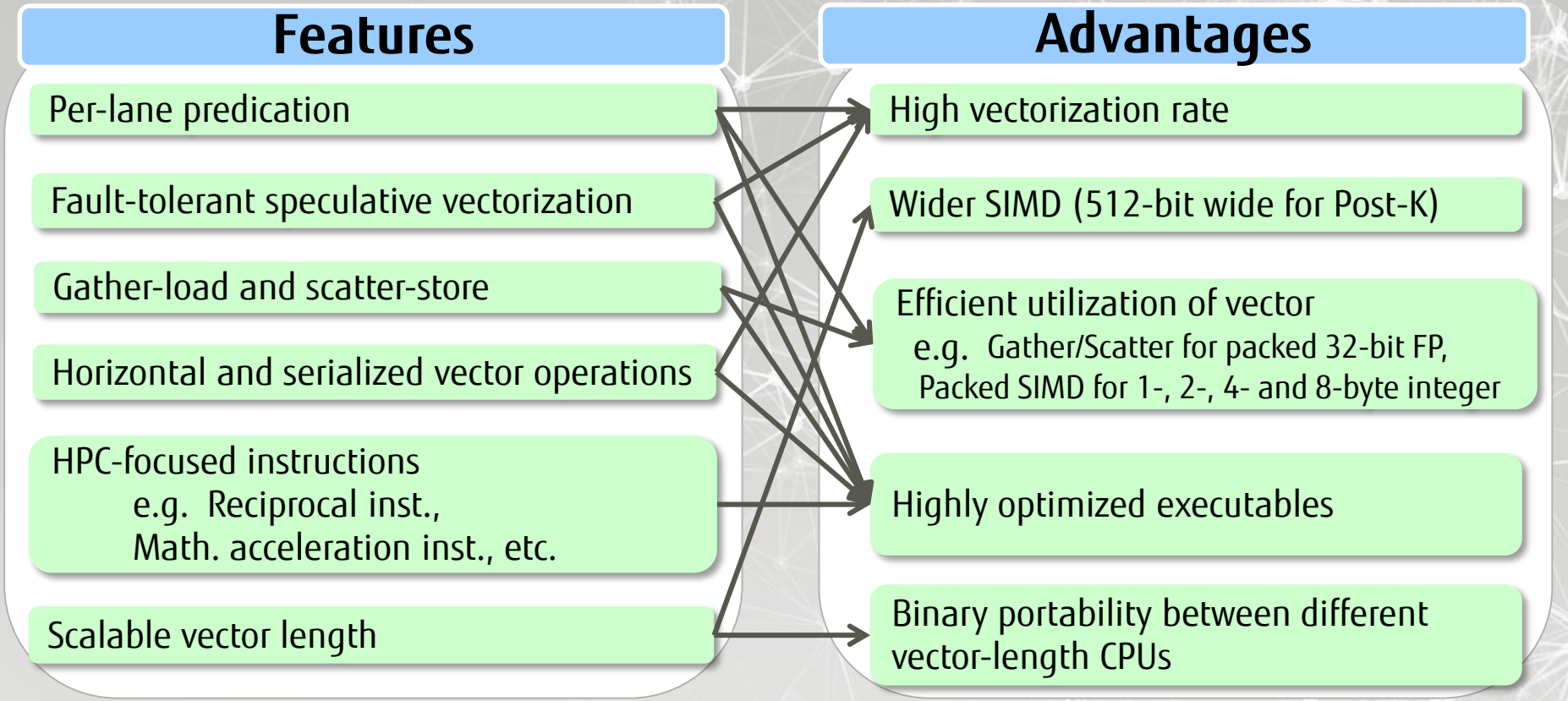
FUJITSU

- The SIMD extension is a 512-bit wide implementation of SVE
- SVE is an HPC-focused SIMD instruction extension in AArch64
  - Co-developed with ARM, taking advantage of Fujitsu's HPC technologies
  - SVE and Advanced SIMD(NEON) are available, concurrently
- FUJITSU's microarchitecture and compiler technologies maximize the execution performance of the Post-K CPU with SVE



SVE's Features and Advantages

FUJITSU



Additional Vectorization Targets in Post-K

FUJITSU

- Post-K targets new kinds of loops
  - Predicate register and fault-tolerant speculative load enable additional vectorization
  - Vectorization of non-"SVE-specific" loops is improved, as well

Vectorization targets as before

```
// Loop including if-statements
for (int i=0; i<n; ++i) {
   if (mask[i] !=0) {  a[i] = b[i];   }
}
```

```
// Short-iteration Loop
for (int i=0; i<VL/2; ++i) {
  a[i] = b[i] * c[i];
}
```

and loops vectorized in K computer and FX100

Additional vectorization targets

```
// While Loop
do {
   b[i] = a[i];
} while(a[i++] != 0);
```

```
// Data-dependent loop
for (int i=0;i<n;++i) {
   a[b[i]] = a[i];
}
```

etc.

Vectorization of While-loop

FUJITSU

- While-loop is vectorized with fault-tolerant speculative instructions
  - Reduces the # of iterations by using SIMD instructions and improves the execution performance of the while-loop

Target loop

```
int a[n], b[n];
...
do {
    b[i] = a[i];
} while(a[i++] != 0);
```

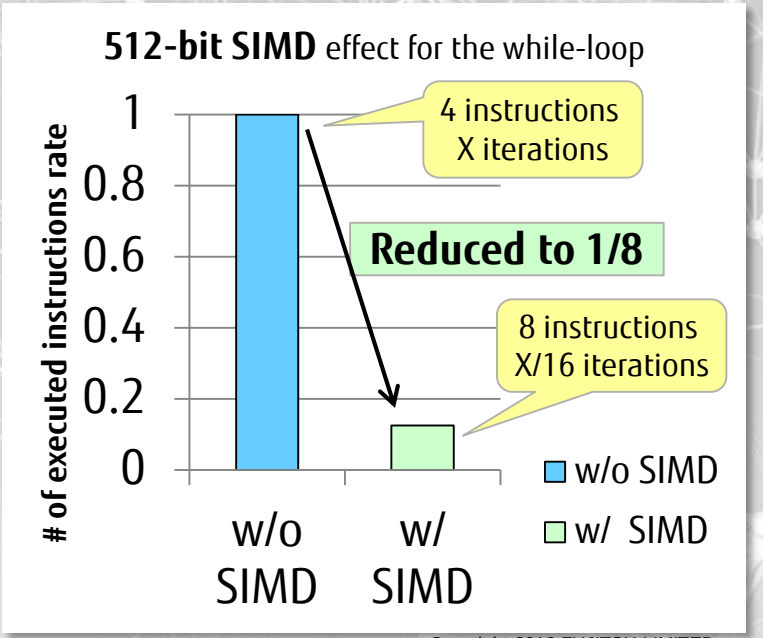
Scalar code w/o SIMD

```
.loop:
   ldr     w1, [x0,x3]  // load a[i]
   str     w1, [x0,x2]  // store b[i]
   add     x0, x0, 4    // i++
   cbnz    w1, .loop    // a[i++] != 0
```

Vector code w/ SVE

```
   setffr
   ptrue   p7.s, all
.loop:
   ldff1w  z0.s, p7/z, [x0, x2, LSL #2]
   rdffr   p0.b, p7/z
   cmpeq   p1.s, p0/z, z0.s, #0
   brkbs   p1.b, p0/z, p1.b
   st1w    z0.s, p1/z, [x1, x2, LSL #2]
   incp    x2,   p1.s
   b.last  .loop

// paths treating faults is omitted
```



Powerful Gather/Scatter for Indirect Array Accesses

FUJITSU

- Accelerates indirect array accesses
  - Supports gather/scatter for 32- and 64-bit FP, and 1-,2-,4- and 8-byte integers
  - Post-K compiler optimizes and vectorizes indirect accesses, making use of the 512-bit vector registers to the greatest extent possible

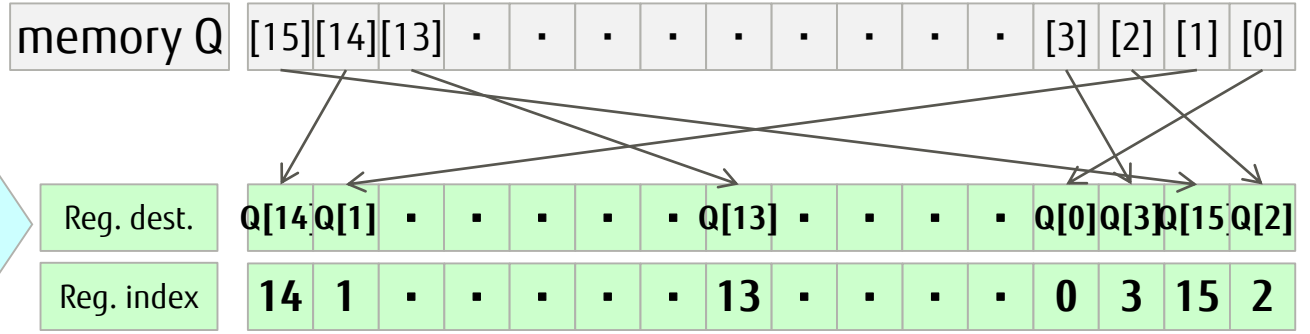
Sample code of array access

```
int index[n]
float P[n], Q[n];

for (i=0; i<n; ++i)
{
  P[i] = Q[index[i]];
}
```

SIMD

Gathered 16 elements of 32bit-FP with 1 instruction



Integer SIMD Utilizing Vector Register

FUJITSU

- Supports SIMD instructions for 1-, 2-, 4- and 8-byte Integers
  - Vectorizes integer operations by utilizing vector registers to the greatest extent as possible

The max number of elements for SIMD

Element Type		HPC-ACE2 (256-bit)	SVE (as 512-bit)
INT	8-byte	4	8
	4-byte	4	16
	2-byte	x	32
	1-byte	x	64
FP	8-byte	4	8
	4-byte	8*	16

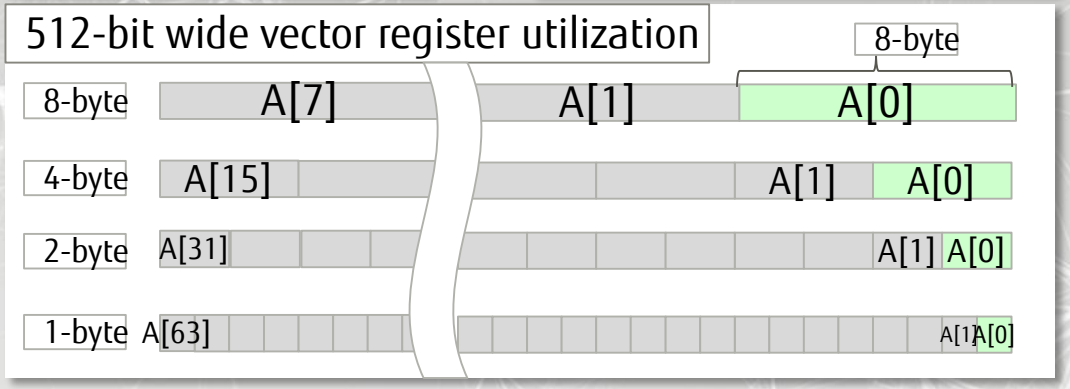
* Only some fundamental operations are available

```
char A[n], B[n], C[n];
for(int i=0;i<64;++i) {
    A[i] = B[i] + C[i];
}
```

SIMD

64-way SIMD

```
ld1b    z1.b, p0/z, [x1]     //  "B"
ld1b    z2.b, p0/z, [x2]     //  "C"
add     z1.b, p0/m, z2.b, z0.b
st1b    z1.b, p0,   [x0]     //  "A"
```



4-Operand Multiply and Add with MOVPRFX

FUJITSU

- SVE supports 3-operand Multiply and Add instructions of 32- and 64-bit FP and 8-, 16-, 32- and 64-bit integers as SIMD
  - FMLA is floating-point Multiply and Add with predication - overwrites the addend Z0

```
FMLA    Z0.D, P0/M, Z1.D, Z2.D  // Z0 ← Z0+Z1*Z2 with P0
```

- 4-operand Multiply and Add in Post-K CPU by prefixed MOVPRFX
  - The prefixed MOVPRFX, copying the source operand Z0 to Z3, does not spend any out-of-order resources by combining the following instruction
  - All source operands Z0, Z1 and Z2 are preserved, while the destination operand Z3 is updated

```
MOVPRFX Z3.D, P0/M, Z0.D          // Z3 ← Z0
FMLA    Z3.D, P0/M, Z1.D, Z2.D    // Z3 ← Z3+Z1*Z2 with P0
```

Advantages of the Post-K Compiler

FUJITSU

- Maximizes the execution performance of HPC applications
  - Covers a wide range of applications, including integer calculations
- Targets 512-bit wide vectorization as well as vector-length-agnostic
  - Fixed-vector-length facilitates optimizations such as constant folding
- Inherits options/features of K computer, PRIMEHPC FX10 and FX100
- Language Standard Support
  - Fully supported : Fortran 2008, C11, C++14, OpenMP 4.5
  - Partially supported : Fortran 2015, C++1z, OpenMP 5.0
- Supports ARM C Language Extensions (ACLE) for SVE
  - ACLE allows programmers to use SVE instructions as C intrinsic functions

// C intrinsics in ACLE for SVE

```
svfloat64_t z0 = svld1_f64(p0, &x[i]);
svfloat64_t z1 = svld1_f64(p0, &y[i]);
svfloat64_t z2 = svadd_f64_x(p0, z0, z1);
svst1_f64(p0, &z[i], z2);
```

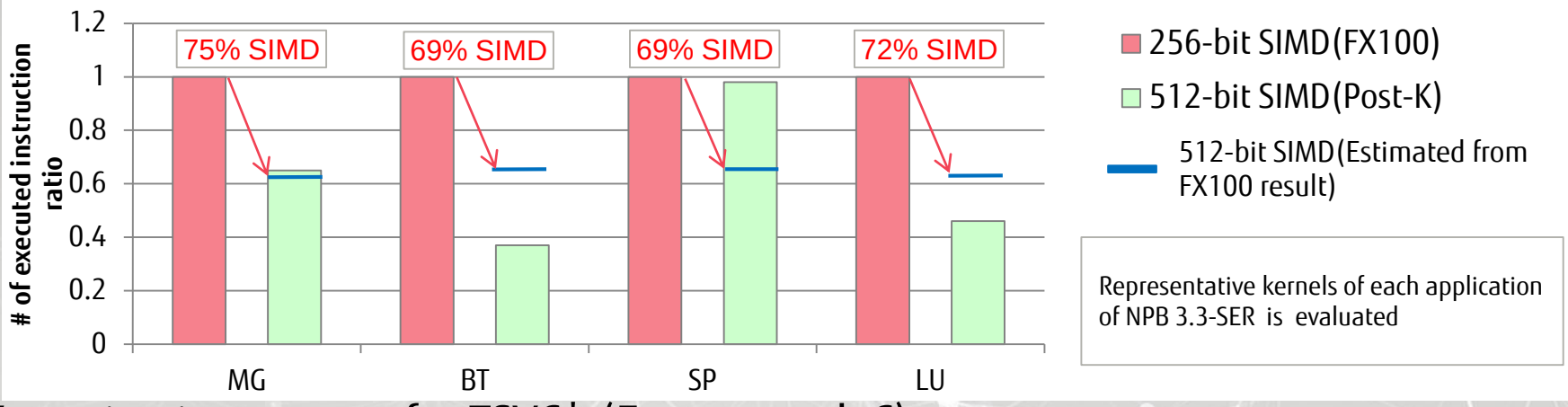
// SVE assembler

```
ld1d    z1.d, p0/z, [x19, x3, lsl #3]
ld1d    z0.d, p0/z, [x20, x3, lsl #3]
fadd    z1.d, p0/m, z1.d, z0.d
st1d    z1.d, p0,   [x21, x3, lsl #3]
```

Auto-vectorization Ability of FUJITSU Compiler for Post-K

FUJITSU

- The # of instructions of NAS Parallel Benchmark measured on FUJITSU's simulator
  - The Post-K compiler is now comparable with the proven FX100, and is improving



- Vectorization rate up for TSVC* (Fortran and C)

TSVC	PRIMEHPC FX100	Post-K Goal
Fortran (135)	96	111 (FX100+15)
C(151)	106	121 (FX100+15)

```
//s482 is one of the vectorized loops thanks to SVE
for (int i = 0; i < LEN; i++) {
  a[i] += b[i] * c[i];
  if (c[i] > b[i]) break;
}
```

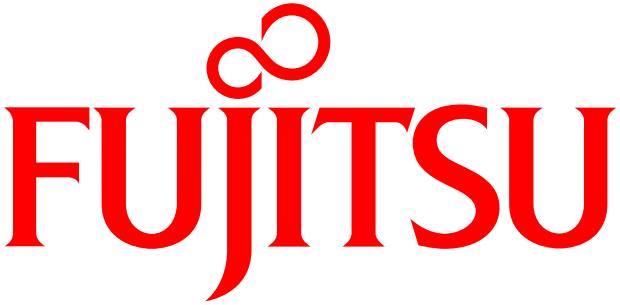
*[Fortran] D. Callahan, J. Dongarra, and D. Levine. "Vectorizing compilers: a test suite and results." In Supercomputing '88, pp. 98- 105.
[C] S. Maleki, Y. Gao, M. J. Garzar´n, T. Wong, and D. A. Padua, "An Evaluation of Vectorizing Compilers," PACT2011, pp. 372-382.

Post-K Improves upon Features of K computer and FX100

FUJITSU

- FUJITSU's microarchitecture and compiler technologies maximize the execution performance of the Post-K CPU with SVE

ISA Features of FX100	Post-K	SVE Instructions and Details
Operation mask (predication)	✓ Enhanced	Almost all of the instructions are predicated
Fault-tolerant instructions	✓ Enhanced	First-fault load and non-fault load
Gather-load/Scatter-store (indirect load/store)	✓ Enhanced	Gather/Scatter for floating-point and integer
Horizontal and serialized vector operations	✓ Enhanced	ADDV, EORV, PNEXT, etc.
Floating-point FMA and Integer FMA	✓ Enhanced	FMAD, FMLA, MAD, MLA, etc.
Packed SIMD	✓ Enhanced	Packed/Unpacked SIMD for integer and floating-point are available
Element permutation and shuffles	✓Enhanced	TBL, EXT, SPLICE, COMPACT, etc.
Reciprocal approx., trigonometric/exponential func.	✓	Inherits HPC-ACE2 as FRECPX, FTMAD, FEXPA, etc.
Stride-load/store	✓	Combination of structure load/store, gather/scatter and element permutations
Sector cache	✓Enhanced	FUJITSU's extension using ARMv8-A tagged addressing



shaping tomorrow with you