FUJITSU

Chapter 1
Hardware

FUJITSU LIMITED

April 2016

Copyright 2016 FUJITSU LIMITED

Contents

FUJITSU

- Hardware
  - Hardware Configuration
  - SPARC64™ XIfx Chip Overview
  - Tofu Interconnect 2
  - SPARC64™ XIfx Specifications
- Hardware Features
  - Assistant Core
  - HPC-ACE2
  - VISIMPACT
  - Register Extensions
  - Arithmetic Function Auxiliary Instructions
  - Sector Cache
- SIMD
  - SIMD Specifications
  - SIMD Technologies

Copyright 2016 FUJITSU LIMITED

FUJITSU

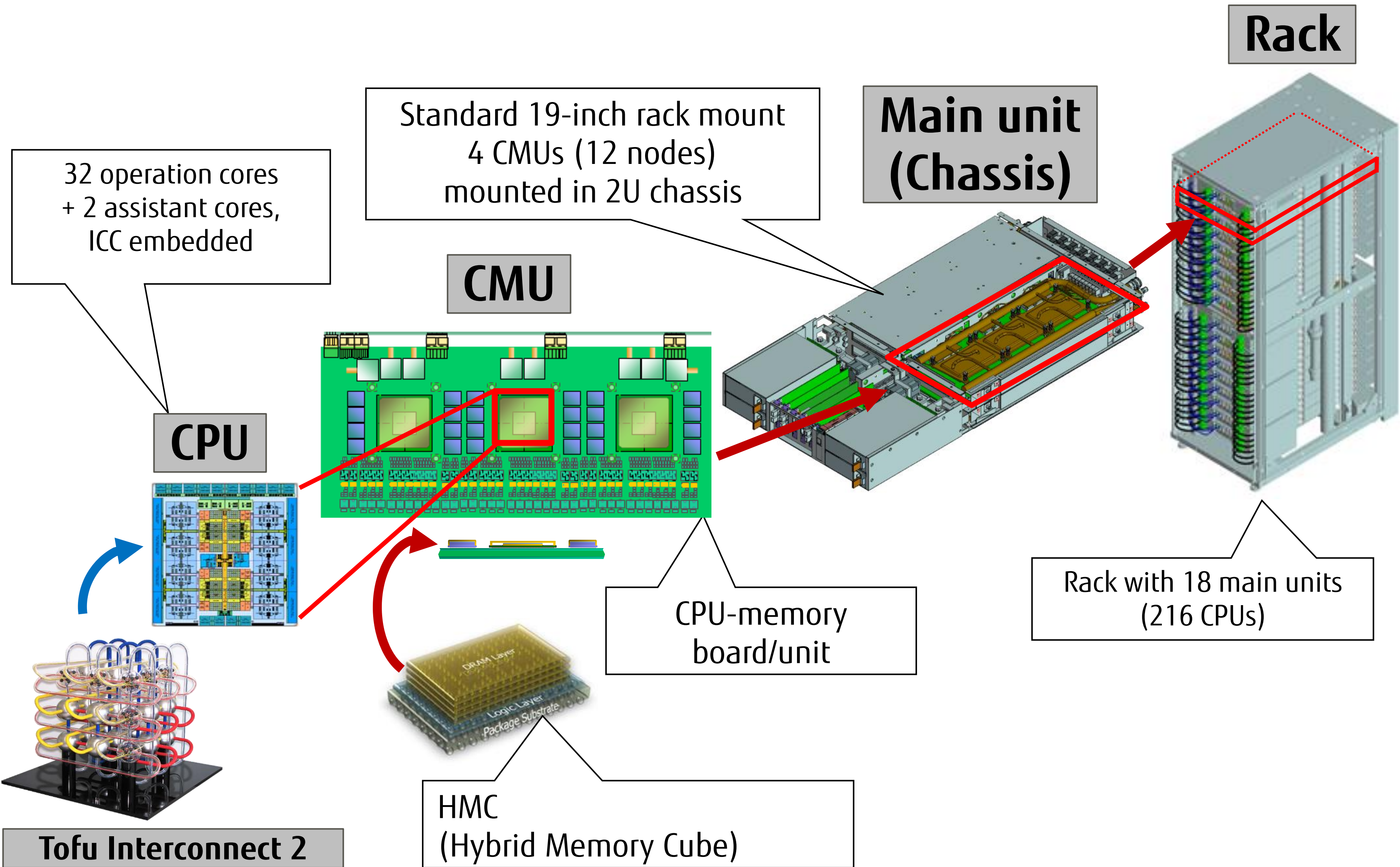
Hardware

- Hardware Configuration
- SPARC64™ XIfx Chip Overview
- Tofu Interconnect 2
- SPARC64™ XIfx Specifications

Copyright 2016 FUJITSU LIMITED

Hardware Configuration

FUJITSU

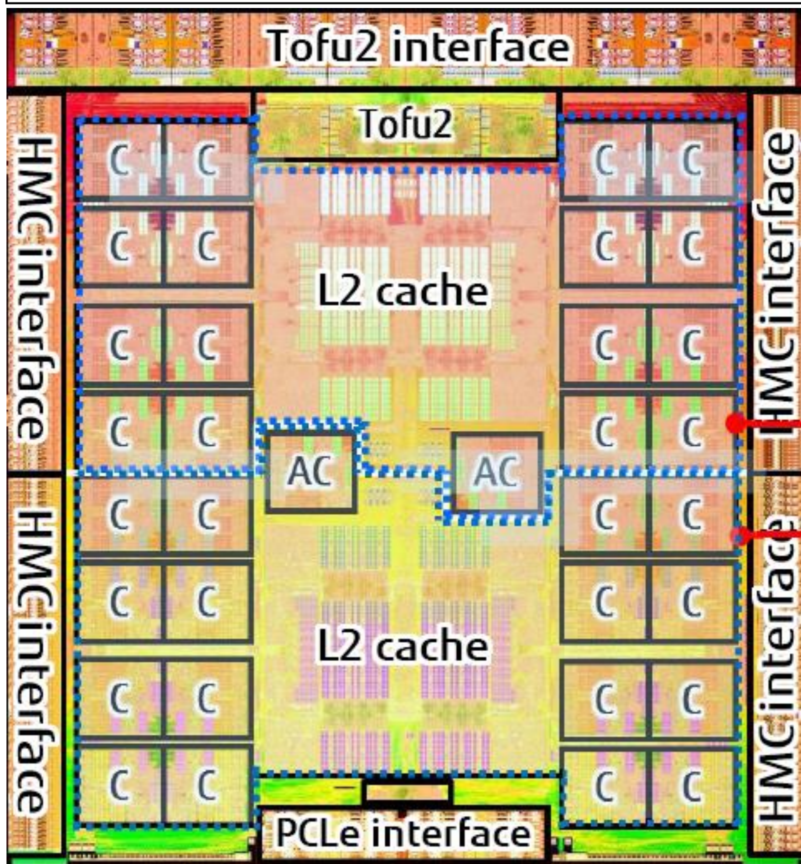


Copyright 2016 FUJITSU LIMITED

SPARC64™ XIfx Chip Overview

FUJITSU

High performance and **high reliability**



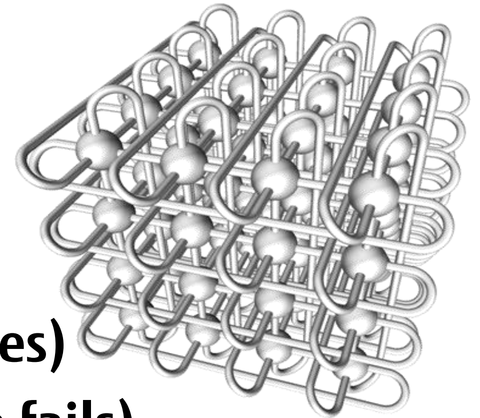
SPARC64™ XIfx

- Architecture
  - 32 operation cores + 2 assistant cores
  - 24 MB shared L2 cache
  - ICC (Inter Connect Controller) embedded
  - 8 HMCs (Hybrid Memory Cubes)
- CMG (Core Memory Group)
  - 16 operation cores + 1 assistant core
  - 12 MB shared L2 cache
  - 2 CMGs per chip
- 20 nm CMOS
  - Number of transistors: Approx. 3.75 billion
  - Number of signal pins: 1001
- Peak performance
  - Operation performance: 1 TFLOPS or more
  - Memory throughput: 480 GB/s

Copyright 2016 FUJITSU LIMITED

Tofu Interconnect 2

FUJITSU



Architecture

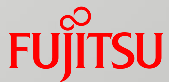
- 6-dimensional mesh/torus
- Direct network with no external switch

Features

- Scalability (supported size of more than 100,000 nodes)
- Fault tolerance (operation can continue when a node fails)
- Communication performance
  - 12.5 GB/s per link, bidirectional
  - Acceleration by optical transmission
    (Optical transmission by all links between the main units)
  - Atomic Read Modify Write supported for RDMA communication
  - High-speed barrier and collective communication(reduction operation)

Copyright 2016 FUJITSU LIMITED

SPARC64™ XIfx Specifications



	SPARC64™ XIfx	SPARC64™ IXfx	Acceleration function
Number of cores	32 + 2	16	Improved parallel processing efficiency (because of addition of assistant cores)
Instruction set	HPC-ACE2	HPC-ACE	Improved instruction-level parallelism
Number of FP registers	128 x 4	128 x 2	
Double-precision operation performance	CPU clock : 1.975GHz 1011.2GFLOPS per chip	CPU clock : 1.88GHz 236 GFLOPS per chip	
	CPU clock : 1.975GHz 1126.4GFLOPS per chip		
Single-precision operation performance	CPU clock : 1.975GHz 2022.4GFLOPS per chip	CPU clock : 1.88GHz 236 GFLOPS per chip	
	CPU clock : 1.975GHz 2252.8GFLOPS per chip		
Number of double-precision operations executed per clock and core	16	8	Doubles the number of concurrent-execution operation instructions.
Double-precision operation unit configuration	2 FMA x 4 SIMD	2 FMA x 2 SIMD	
Single-precision operation unit configuration	2 FMA x 8 SIMD	2 FMA x 2 SIMD	
L1 cache	Specifications by core: Instruction: 64 KB/4WAY Data: 64 KB/4WAY	Specifications by core: Instruction: 32 KB/2WAY Data: 32 KB/2WAY	
L2 cache	24 MB (Shared by cores: 12 MB/CMG)	Shared by cores: 12 MB	

Copyright 2016 FUJITSU LIMITED

FUJITSU

Hardware Features

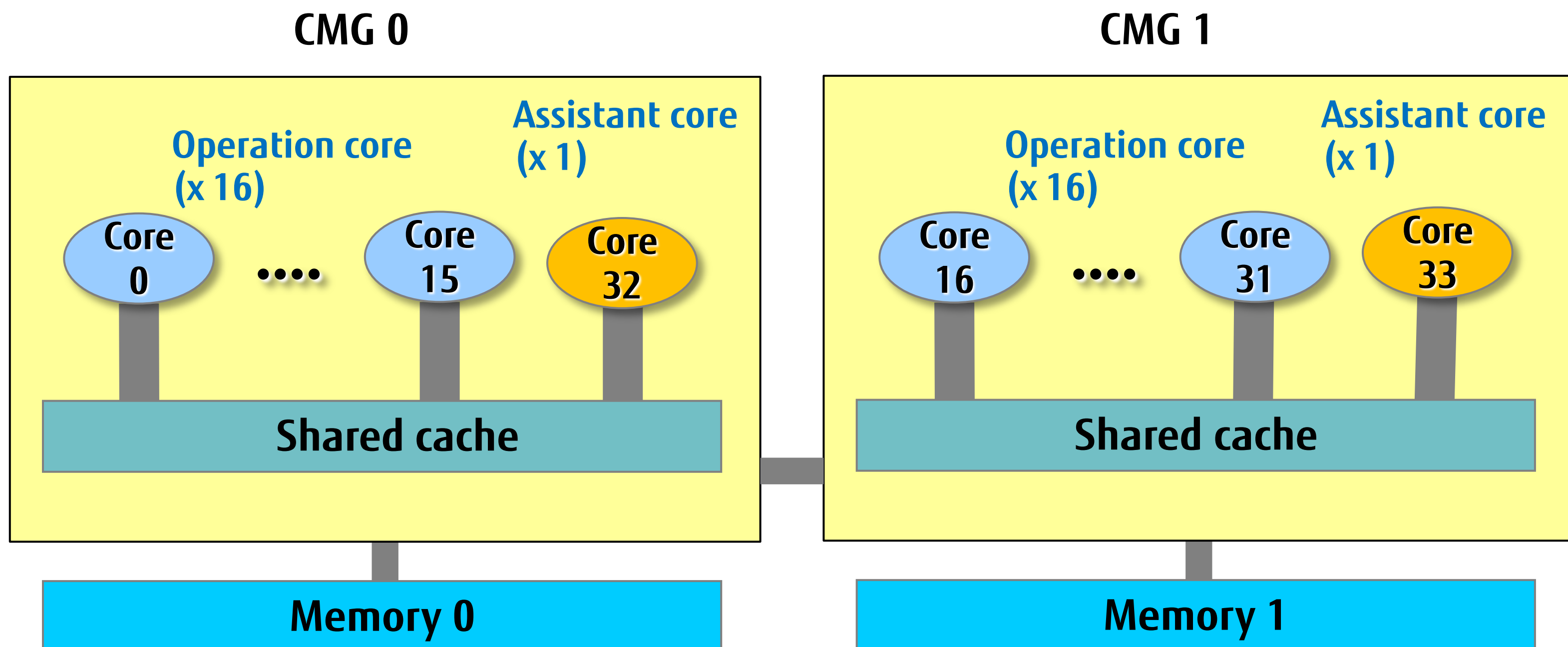
- Assistant Core
- HPC-ACE2
- VISIMPACT
- Register Extensions
- Arithmetic Function Auxiliary Instructions
- Sector Cache

Copyright 2016 FUJITSU LIMITED

Assistant Core

FUJITSU

- Two assistant cores are installed for 32 operation cores.
- The cores are not used by user applications but are responsible for OS processing, etc.
- Purposes of use
  - OS noise reduction
  - Overlapping execution of operation and communication
  - Routing of IO data (between Tofu and InfiniBand)



Copyright 2016 FUJITSU LIMITED

HPC-ACE2

FUJITSU

(High Performance Computing - Arithmetic Computational Extensions 2)

- SPARC64™ Xifx ISA (Instruction Set Architecture)
  - Compliance specifications
    - SPARC-V9 specifications
    - JPS (Joint Programmer's Specification): SPARC-V9 extended specifications
  - HPC-ACE2: Second generation of HPC-ACE, which is Fujitsu's proprietary extended instruction set for HPC
    - Floating-point register extension
    - Sector cache
    - Arithmetic function auxiliary instructions
    - SIMD (Single Instruction Multiple Data) instructions
    - Stride SIMD load and store instructions
    - SIMD indirect instructions
    - VISIMPACT

Copyright 2016 FUJITSU LIMITED

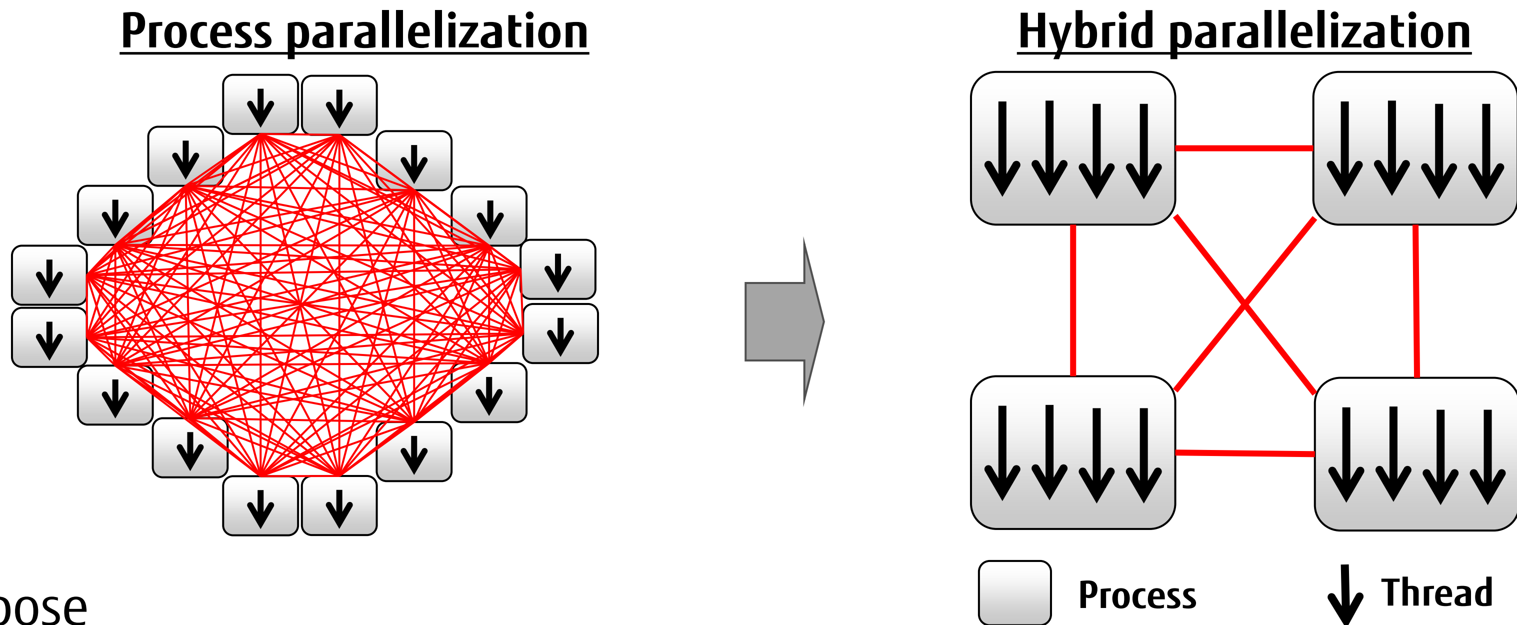
VISIMPACT - Mechanism Facilitating Hybrid Parallelization -
(Virtual Single Processor by Integrated Multi-core Parallel Architecture)

FUJITSU

Concept

- **Hardware barrier function realizing highly efficient thread parallelization processing within a multi-core CPU**
- **Mechanism helping to realize a highly efficient hybrid parallelization execution model**

MPI + thread parallelization processing (automatic parallelization/OpenMP)



Purpose

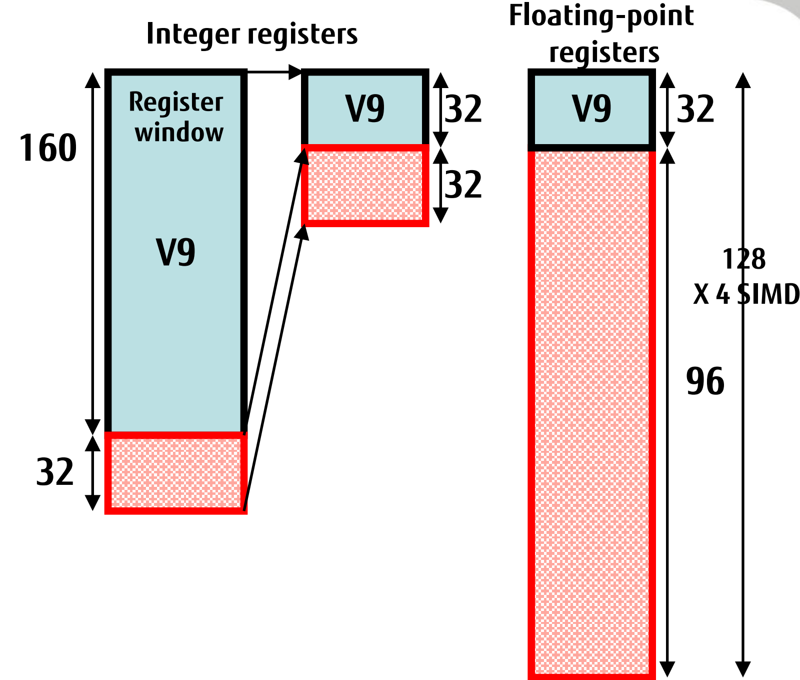
- **You can achieve following effects by treating a multi-core CPU as a single high-speed CPU**
  - reduces the number of MPI processes to 1/n cores
  - improves the efficiency of thread parallelization processing
  - reduces memory access

Copyright 2016 FUJITSU LIMITED

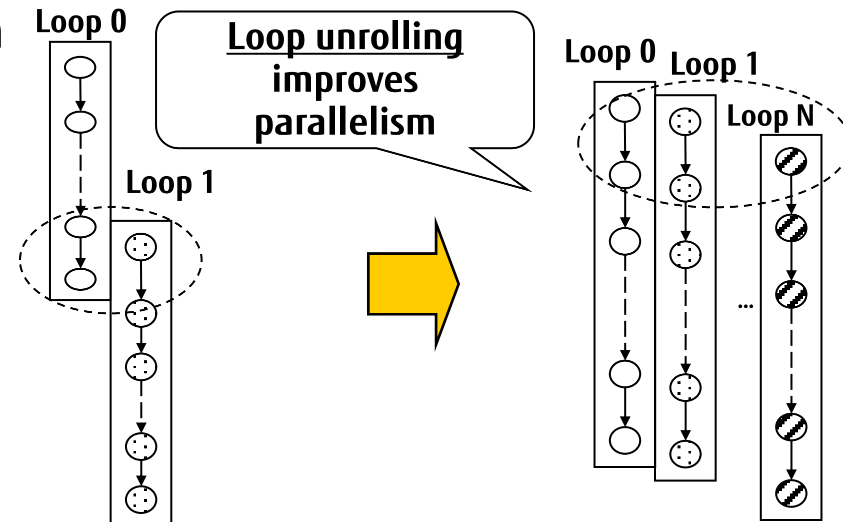
Register Extensions

FUJITSU

- Registers extended from V9
  - Integer registers: 32 ⇨ 64
  - Double-precision floating-point registers: 32 ⇨ 128 x 4 SIMD
    - The floating-point registers are all the same.
    - The extended floating-point registers are also accessible from non-SIMD instructions.



- Extension reasons
  - To improve instruction-level parallelism, which was limited by the number of registers
  - To reduce the overhead due to register spill/fill



* Spill/Fill

Spill/Fill refers to the operation of temporarily saving data to memory when the registers needed during an operation are insufficient, and then restoring the data.

Copyright 2016 FUJITSU LIMITED

Arithmetic Function Auxiliary Instructions

FUJITSU

- Extension of arithmetic function auxiliary instructions
  - Instructions that assist the approximation of trigonometric functions (sin and cos)
  - Instructions for the approximation of reciprocals to accelerate division and square root calculations
- Auxiliary instructions added in HPC-ACE2 and later
  - Auxiliary instructions for exponential functions
  - Rounding operations instructions

Copyright 2016 FUJITSU LIMITED

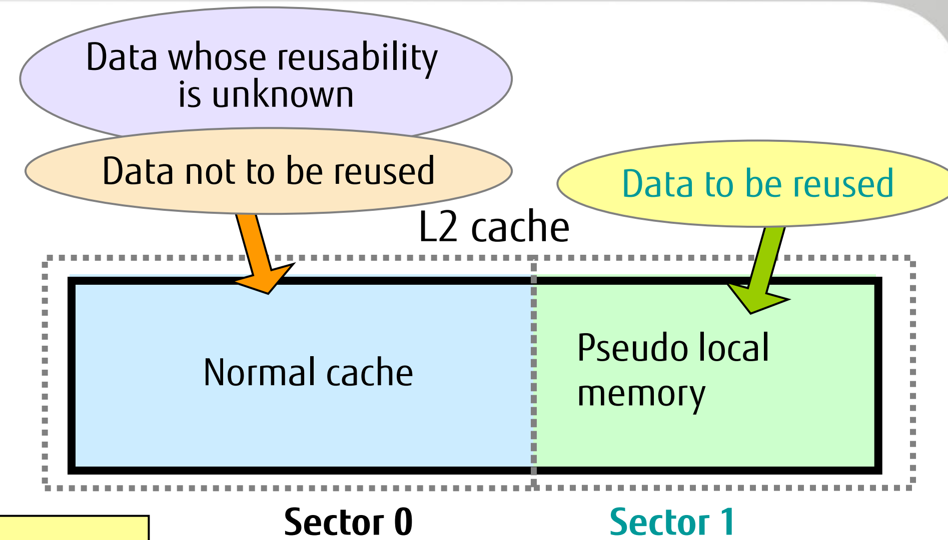
Software-controlled Cache: Sector Cache

FUJITSU

- **Sector cache: Pseudo local memory**

⇨ Software can use sectors effectively according to the reusability of data.

  - Arrays for reuse ⇨ Sector 1 used
  - Others ⇨ Sector 0 used
  - Data on sector 1 is not forced out by other data.
  - The user can specify in a directive line that the array be in sector 1.



<Purpose>
To prevent array a from being forced out of the cache by access to array b and array c in a loop.

```
!OCL CACHE_SECTOR_SIZE(15,9)
!OCL CACHE_SUBSECTOR_ASSIGN(a)
do j=1,m
   do i=1,n
      a(i) = a(i) + b(i,j) * c(i,j)
   enddo
enddo
!OCL END_CACHE_SUBSECTOR
!OCL END_CACHE_SECTOR_SIZE
```

Example of using a compiler directive line to specify a sector cache

SIMD

- SIMD Specifications
- SIMD Technologies

Copyright 2016 FUJITSU LIMITED

SIMD Specifications (1/4)

FUJITSU

- SIMD (Single Instruction Multiple Data)
  - A single instruction performs multiple operations.
  - SIMD registers are used to manipulate multiple data.
  - Instruction-level parallelism within a core is improved.
- SIMD instructions
  - Instructions that use SIMD registers
  - Operation instructions and memory access instructions
- How SIMD instructions apply acceleration
  - Optimization by a compiler (SIMD optimization)
  - Programming with intrinsic functions

Multiple operations by single SIMD instruction

```
fadd,s f2 f4 f6
a(i)   = b(i)   + c(i)
a(i+1) = b(i+1) + c(i+1)
a(i+2) = b(i+2) + c(i+2)
a(i+3) = b(i+3) + c(i+3)
```

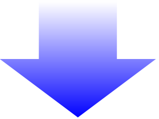

Copyright 2016 FUJITSU LIMITED

SIMD Specifications (2/4)

FUJITSU

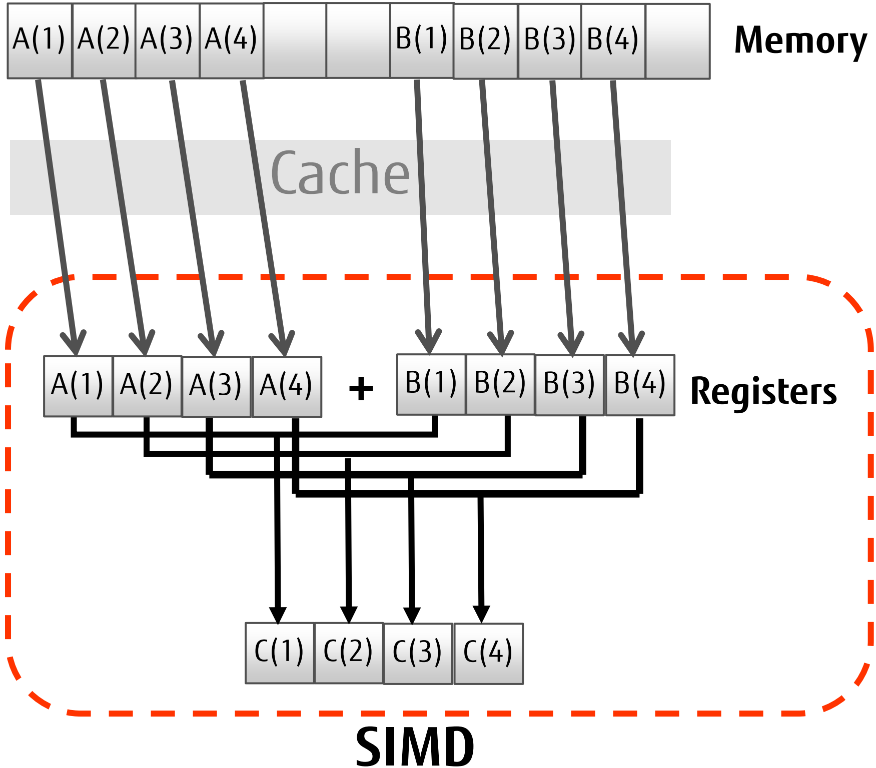
- Principles of SIMD optimization by the compiler
  - Analyzing the direction of loop rotation and the similarity between instruction sequences
  - Outputting SIMD instructions according to the application scenario

```
do i=1,4
  c(i) = a(i) + b(i)
enddo
```



```
ldd,s a(i:i+3) f2
ldd,s b(i:i+3) f4
fadd,s f2 f4 f2
store,s f2 c(i:i+3)
```

=



Copyright 2016 FUJITSU LIMITED

SIMD Specifications (3/4)

FUJITSU

- SIMD optimization by the compiler
  - VSIMD (Vectorize SIMD)
  - UXSIMD (Un-loop eXaminate SIMD)

- Hardware specifications for SIMD in the FX100
  - 256-bit wide SIMD
    - For details, see "SIMD Technologies (2/11)."
  - Up to eight operations are processed simultaneously per SIMD instruction (single precision).
  - Two SIMD operation instructions are performed simultaneously within a core.

Copyright 2016 FUJITSU LIMITED

SIMD Specifications (4/4)

FUJITSU

Comparison of SIMD instruction specifications and functions

	FX10	FX100
Double-precision real (operation instruction, load and store instruction)	2 SIMD	2 SIMD, 4 SIMD
Single-precision real (operation instruction, load and store instruction)	2 SIMD	2 SIMD, 4 SIMD, 8 SIMD
Integer (operation instruction, load and store instruction)	-	2 SIMD, 4 SIMD
Masked instruction	Real	Real, integer
Data boundary (load)	Type size	Type size
Data boundary (store)	Type size x 2	Type size
Non-sequential data access instruction (stride)	-	Load and store
Non-sequential data access instruction (indirect)	-	Load, store, prefetch

Copyright 2016 FUJITSU LIMITED

SIMD Technologies (1/11)

FUJITSU

Representative SIMD technologies

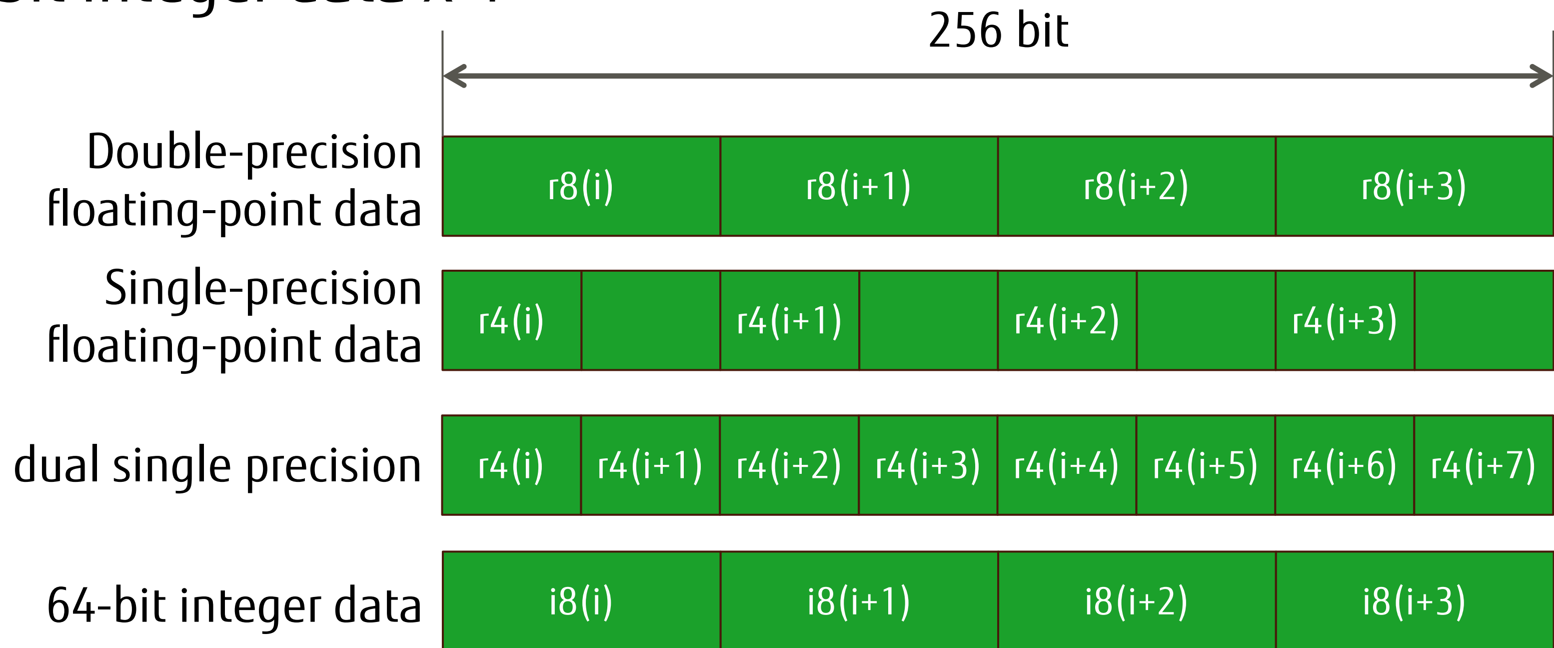
- Technology list
  1. 256 bit wide SIMD
  2. Stride memory access
  3. Indirect memory access
  4. Loop Fission
  5. Unaligned SIMD store
  6. Integer condition branch
  7. New complex-number model
  8. Concatenation shift
  9. Element compress
  10. Masking loop SIMD

Copyright 2016 FUJITSU LIMITED

SIMD Technologies (2/11)

FUJITSU

- 256 bit wide SIMD
  - Double-precision floating-point data x 4
  - Single-precision floating-point data x 4, dual single-precision floating-point data x 4
  - 64-bit integer data x 4



The SIMD width is 2 and 4 times wider in double and single precision, respectively, compared with the FX10.

Copyright 2016 FUJITSU LIMITED

SIMD Technologies (3/11)

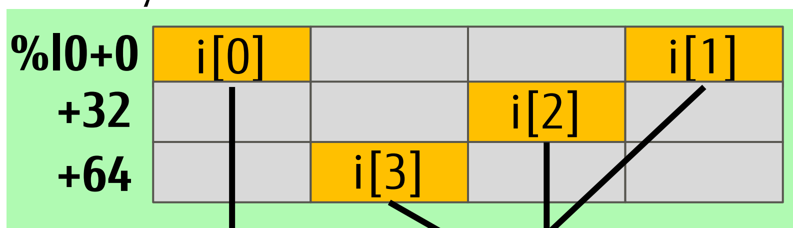
FUJITSU

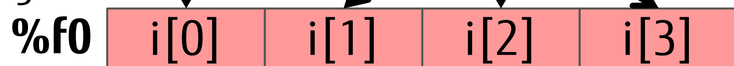
Stride memory access

- Applicable to stride lengths of 2 to 7

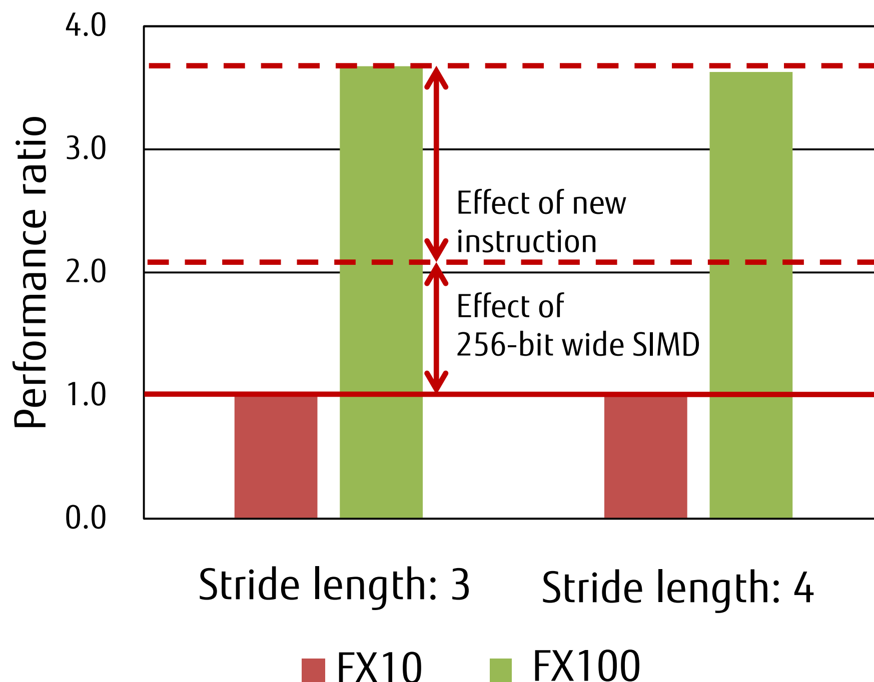
Load instruction with stride length of 3

Memory





Stride load performance (1 core)



Effective performance in stride access to continuum code, etc.

Copyright 2016 FUJITSU LIMITED

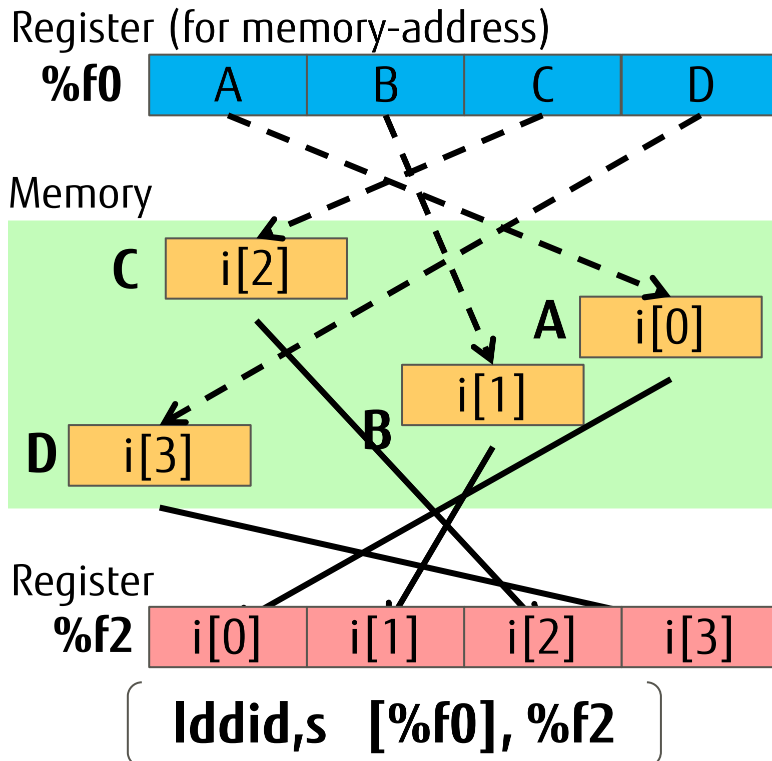
SIMD Technologies (4/11)

FUJITSU

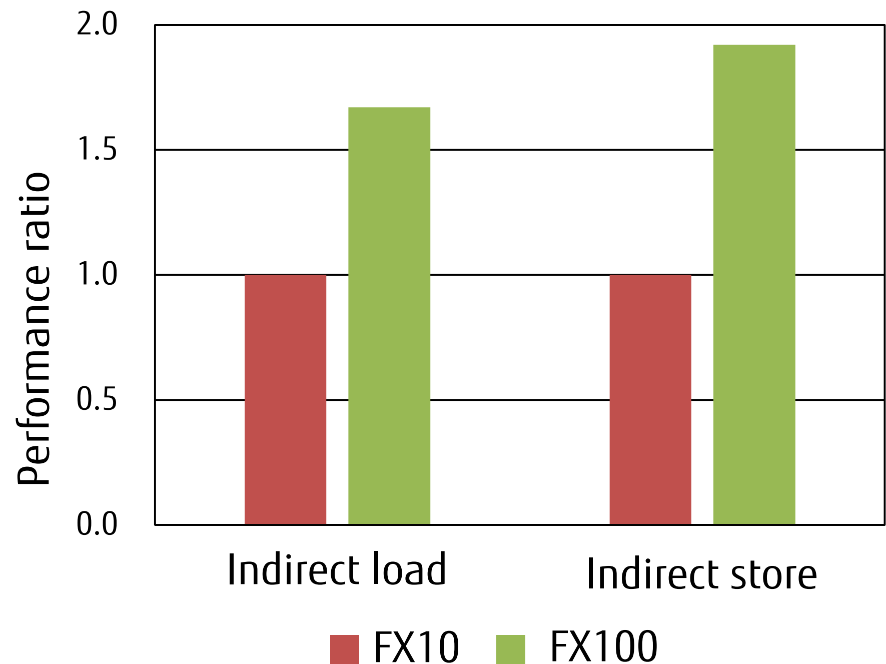
Indirect memory access

- Address calculation also uses SIMD instructions for parallel computation.

Indirect load instruction



Indirect access performance (1 core)



Effective performance in list access, such as for fluid analysis or FEM

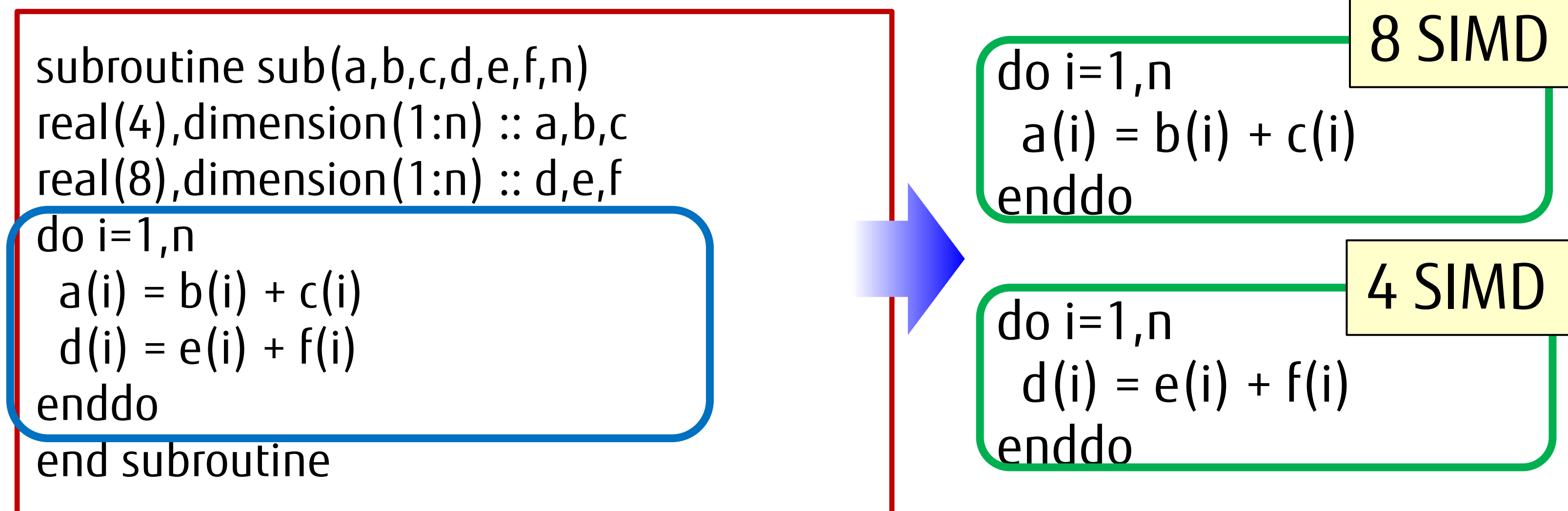
Copyright 2016 FUJITSU LIMITED

SIMD Technologies (5/11)

FUJITSU

Loop Fission

- Mix of single- and double-precision floating-point operations
- Loop fission resulting in different SIMD widths



For a loop containing a mix of single- and double-precision floating-point operations, the result will be 4 SIMD.

To turn the single-precision floating-point operations into 8 SIMD, divide the loop by different SIMD widths.

Effective use of 256-bit wide SIMD

Copyright 2016 FUJITSU LIMITED

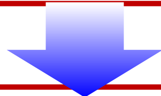
SIMD Technologies (6/11)

FUJITSU

Unaligned SIMD store

- Store instructions are not affected by differences in data boundaries.
- The boundary corrections output by FX10 is eliminated.

```
subroutine sub(a,b,c,n)
do i=1,n
  a(i) = b(i) + c(i)
enddo
end subroutine
```



```
Appearance of FX100 object
i=1
do j=i,n,4
  a(j:j+3) = b(j:j+3) + c(j:j+3)
enddo
...
```

```
Appearance of FX10 object
i=1
if (and(loc(a(1)),0xf) .ne. 0) goto 10
a(1) = b(1) + c(1)
i=2
10 conitnue
do j=i,n,4
  a(j:j+3) = b(j:j+3) + c(j:j+3)
enddo
...
```

Eliminated boundary correction

Reduced code size and more efficient instruction cache

Copyright 2016 FUJITSU LIMITED

SIMD Technologies (7/11)

FUJITSU

Integer condition branch

- SIMD optimization of loops containing IF construct for integer-type condition determination
- FX10 converts data into floating-point type data.

```
real(8),dimension(1:n) :: a,b,c
integer(4),dimension(1:n) :: m
do i=1,n
  if (m(i) .gt. 0) then
    a(i) = b(i) + c(i)
  endif
enddo
```

New instructions for FX100 integer type

```
! integer-type SIMD load
ldsw,s      m(i:i+3),%f32

! integer-type SIMD compare
fzero,s     %f34
fpcmpgtw,s  %f32,%f34,%f36
```

Appearance of conversion instruction output in FX10

```
if (real(m(i:i+1),kind=8) .gt. 0.0) then
   a(i:i+1) = b(i:i+1) + c(i:i+1)
```

Type conversion (8-byte real conversion) not necessary

Elimination of unnecessary conversion instructions and improvement of scheduling

Copyright 2016 FUJITSU LIMITED

SIMD Technologies (8/11)

FUJITSU

New complex-number model

- Instruction output using stride memory access
- SIMD optimization of adjacent memory by FX10

```
subroutine sub(a,b,r,n)
complex(8),dimension(1:n) :: a,b
real(4) :: r
do i=1,n
  a(i) = b(i) * r
enddo
end subroutine
```

Stride memory access used because it is of complex type

New instructions for FX100 instruction output integer type

```
7 instructions in total
ldd      [%o2], %f2  !argument r
lddst,s  b(i:i+3).r, %f4
lddst,s  b(i:i+3).r, %f6
fmuld,s %f4, %f2, %f4
fmuld,s %f6, %f2, %f6
stdst,s  %f4, a(i:i+3).r
stdst,s  %f6, a(i:i+3).i
```

Number of instructions in FX10

11 instructions in total
Load x 4, store x 2, operation x 4, and initialization x 2

Number of instructions reduced to accelerate operations of complex type

Copyright 2016 FUJITSU LIMITED

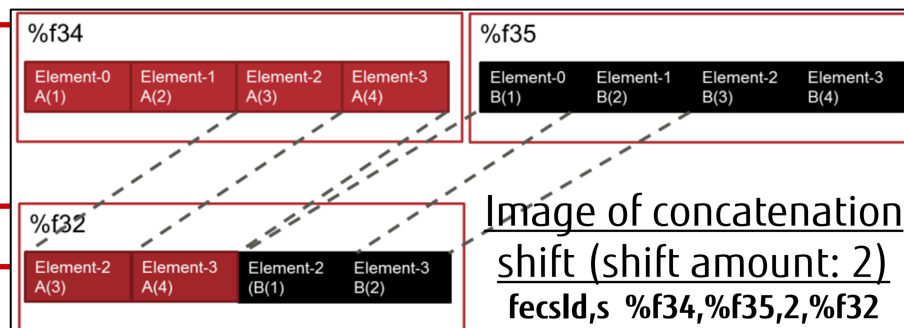
SIMD Technologies (9/11)

FUJITSU

Concatenation shift

Concatenation of memory references in the loop rotation direction

```
do i=1, n
 a(i) = b(i) + b(i+1) + b(i+2) + b(i+3)
enddo
```




```
T1 = b(1:4)
do i=1, n-4, 4
  T2 = b(i+4:i+7)                                     * Preceding LOAD
  T3 = concatenate_shift(T1, T2, 1)   * Converts the LOAD instruction for b(i+1:i+4).
  T4 = concatenate_shift(T1, T2, 2)   * Converts the LOAD instruction for b(i+2:i+5).
  T5 = concatenate_shift(T1, T2, 3)   * Converts the LOAD instruction for b(i+3:i+6)
  T6 = T1 + T3
  T7 = T4 + T5
  a(i:i+3) = T6 + T7
  T1 = T2
enddo
```

Output of only single SIMD load instruction in loop b(i+4:i+7)

Reduced load instructions from adjacent access, such as in stencil code

Copyright 2016 FUJITSU LIMITED

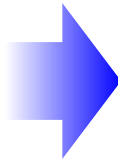
SIMD Technologies (10/11)

FUJITSU

Element compress

- Compression using masks and output of horizontal addition instructions
- Reduced branches in a loop

```
j=1
do i=1,10000
  if (x(I) .gt. 0.d0) then
    a(j)=b(i)
    j=j+1
  endif
enddo
```



Appearance of FX100 instruction output

```
j=1
do i=1,10000,4
  mask = x(i:i+3) .gt. 0d0
  vtd = b(i:i+3)
  cvtd=fecpd(vtd,mask)
  a(j:j+3)=cvtd
  j=j+int(fesummd(mask),kind=4)
enddo
```

fecpd: Mask assignment shown here
fesummd: Mask addition shown here

Acceleration of array compression code, etc.

Copyright 2016 FUJITSU LIMITED

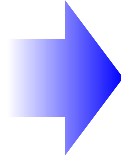
SIMD Technologies (11/11)

FUJITSU

Masking loop SIMD

- SIMD optimization of short loops using mask operations
- Applied using optimization options because operations would otherwise be performed redundantly

```
do i=1,n
  a(i) = b(i) + c(i)
enddo
```



Appearance of FX100 instruction output

```
Loop:
  nn = remaining number of
          rotations/4
  idx = (nn>=1) ?4:nn
  ldd,s  mtbl[idx], %f8
  ldd,s  b(i:i+3), %f2
  ldd,s  c(i:i+3), %f4
  faddd,s  %f2, %f4, %f6
  stdfr,s  %f6, %f8, a(i:i+3)
  branch Loop, cond
```

```
mtbl[0]={F,F,F,F}
mtbl[1]={T,F,F,F}
mtbl[3]={T,T,T,F}
mtbl[4]={T,T,T,T}
```

Acceleration for loops with fewer rotations

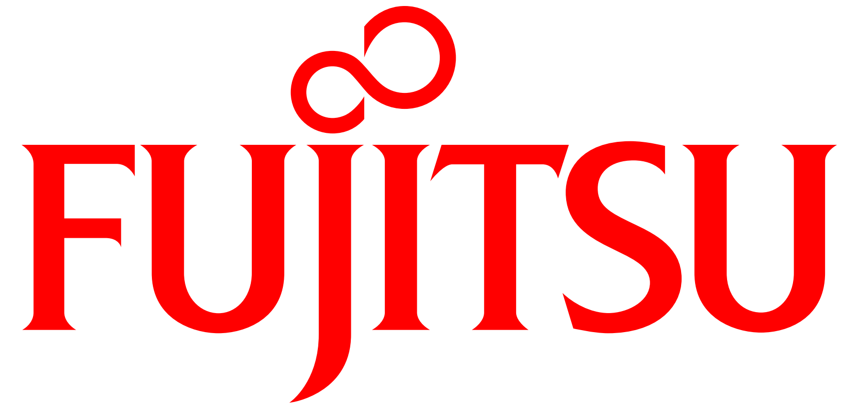
Copyright 2016 FUJITSU LIMITED

Revision History

FUJITSU

Version	Date	Revised section	Details
2.0	April 25, 2016	-	- First published

Copyright 2016 FUJITSU LIMITED



shaping tomorrow with you