Programming Guide (Programming common part)

Mar. 2023

V1.6

FUJITSU LIMITED

Introduction

- This document is intended for application developers and tuning engineers for the A64FX processor.
- Refer to the following in conjunction with this document.
  - Fortran User's Guide
  - C User's Guide
  - C++ User's Guide
  - Profiler User's Guide
  - Programming Guide(Processors)
  - Programming Guide(Tuning)
  - Programming Guide(Fortran)
  - Manuals provided by Technical Computing Suite Job Operation Software
- The following abbreviation is used in this document:
  - A64FX Logic Specifications
  - A64FX ® Microarchitecture Manual
  - ARM® Architecture Reference Manual (ARMv8 , ARMv8.1 , ARMv8.2 , ARMv8.3)
  - ARM® Architecture Reference Manual Supplement The Scalable Vector Extension

- Trademarks
  - Linux® is a trademark or registered trademark of Linus Torvalds in the United States and other countries.
  - Red Hat is a trademark or registered trademark of Red Hat Inc. in the United States and other countries.
  - ARM is a trademark or registered trademark of ARM Ltd. in the United States and other countries.
  - Proper names such as the product name mentioned are trademark or registered trademark of each company.
  - Trademark symbols such as ® and ™ may be omitted from system names and product names in this document.

Programming common part - Contents (1)

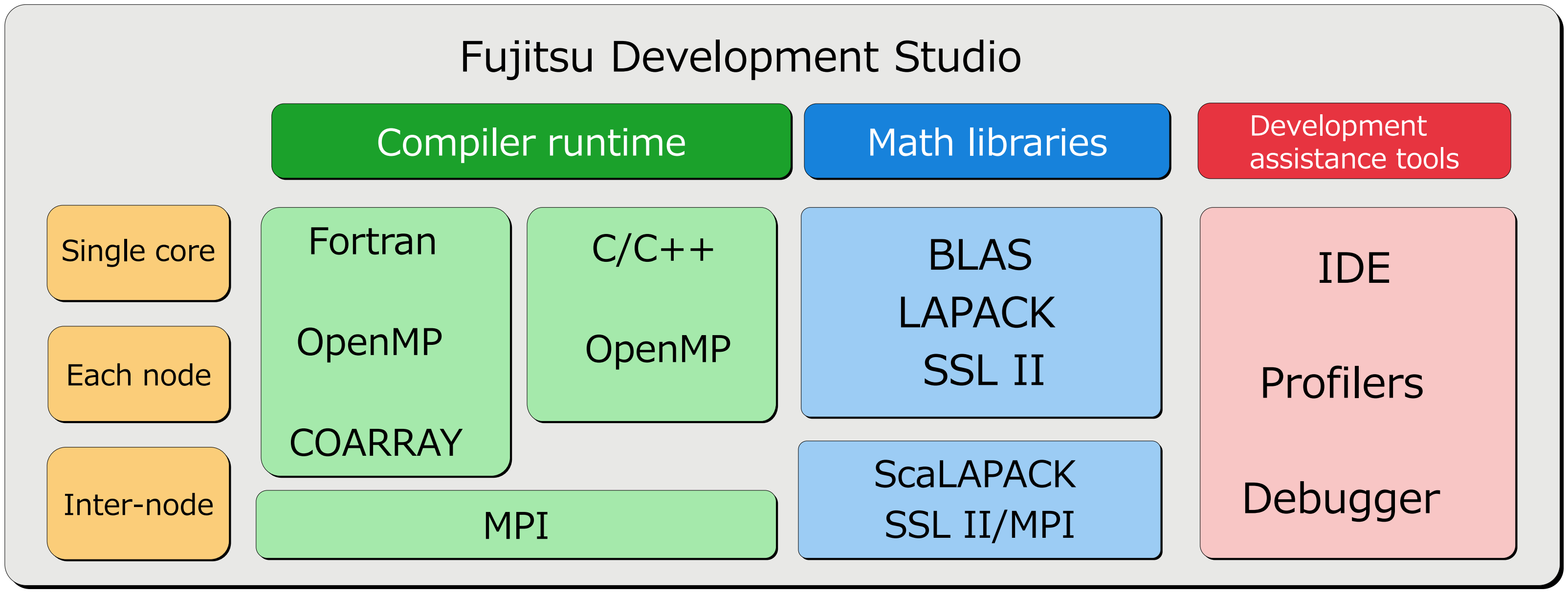
Programming common part - Contents (2)

Programming common part - Contents (3)

Program Development Environment Overview

Program Development Environment System

- Fujitsu Development Studio configuration
  - Fujitsu Development Studio provides a cross compiler that runs on the PRIMERGY and a native compiler that runs on the A64FX. These two types of compiler can support various forms of use.
  - The C/C++ compiler can run in clang mode, which is compatible with Clang/LLVM. By using clang mode, users can easily run open source applications.



Language Specification

- Compliance with the latest standards and industry standard specifications
  - Fujitsu Development Studio complies with the latest standards and industry standard specifications, enabling application development with the latest standards and also making it easy to run open source applications on the A64FX.

Supported Standards

Language	Supported Standards/Specifications
Fortran	Part of ISO/IEC 1539-1:2018 (Fortran 2018 standard) ISO/IEC 1539-1:2010 (Fortran 2008 standard) ISO/IEC 1539-1:2004, JIS X 3001-1:2009 (Fortran 2003 standard) ISO/IEC 1539-1:1997, JIS X 3001-1:1998 (Fortran 95 standard) Fortran 90 and Fortran 77 standards
C	ISO/IEC 9899:2011 (C11 standard) ISO/IEC 9899:1999 (C99 standard) ISO/IEC 9899:1990 (C89 standard) * Extension specifications of the GNU compiler are also supported.
C++	Part of ISO/IEC 14882:2017 (C++17 standard) ISO/IEC 14882:2014 (C++14 standard) ISO/IEC 14882:2011 (C++11 standard) ISO/IEC 14882:2003 (C++03 standard) * Extension specifications of the GNU compiler are also supported.
OpenMP	Part of OpenMP API Version 5.0 OpenMP API Version 4.5
MPI	Part of Message-Passing Interface Standard Version 3.1 and 4.0

MPI Acceleration

- Faster MPI using the Tofu Interconnect D
  - The significant enhancement of communication capabilities from conventional four-way simultaneous communication to six-way simultaneous communication has improved collective communication algorithms (including Allgather and Bcast), which require a wide communication bandwidth for each node.

Interconnect	Number of Network Interfaces	Target Machine
Tofu	4 (1 node)	FX10
Tofu2	4 (1 node)	FX100
Tofu Interconnect D	6 (1 node)	A64FX

- Faster MPI with support for network layers
  - The A64FX has four mounted CMGs (core memory groups) and employs the NUMA (non-uniform memory access) architecture to connect these CMGs through a ring bus.
  - In process parallelization across CMGs, links on the layered network topology are made using the Tofu Interconnect D and the ring bus within the A64FX.
  - The Alltoall algorithm implements an algorithm optimized to minimize communication path conflicts in the ring bus, delivering higher speeds.

Faster Math Libraries (1/2)

- Faster math libraries using internal core speed-up functions
  - Internal core speed-up functions have been applied to the following fast math libraries provided by Fujitsu Development Studio: BLAS and LAPACK are well-known in the linear algebra field; Fujitsu math library SSL II is used by a wide range of R&D users in Japan because its algorithms cover many fields; and the fast quadruple-precision basic math library delivers higher speeds by handling quadruple-precision values in the double-double format.
  - Applications can be faster simply by calling these math libraries.

Sequential Libraries (Thread Safe)	
BLAS, LAPACK	These linear algebra libraries developed in the United States and released by Netlib are de facto standards. BLAS has about 80 routines, and LAPACK has about 400 routines. Some BLAS routines independently support FP16.
SSL II	Library containing about 300 Fortran routines covering a wide range of fields
C-SSL II	C interface for SSL II
Fast quadruple-precision basic math library	This library represents and calculates quadruple-precision values in the double-double format. Some thread parallelization routines are included.

Faster Math Libraries (2/2)

- Faster math libraries using the thread parallelization function
  - Fujitsu Development Studio provides BLAS, LAPACK, and SSL II as fast math libraries having thread parallelization.
  - Applications can be faster with thread parallelization by calling these math libraries.

Sequential Libraries (Thread Safe)	
BLAS, LAPACK	The interface is the same as for the sequential libraries. Also included is PLASMA, which is a task parallel version of LAPACK. They can also be used from Python (NumPy, SciPy).
SSL II thread parallelization function	Thread parallel versions of about 80 important routines. The interface is different from that of the sequential library SSL II, so they can be used concurrently.
C-SSL II thread parallelization function	C interface for the SSL II thread parallelization function

- Faster math libraries using the MPI parallelization function
  - Fujitsu Development Studio provides ScaLAPACK and SSL II/MPI as fast math libraries having MPI parallelization.
  - Acceleration can be faster with MPI parallelization by calling these math libraries.

MPI Parallelization Libraries	
ScaLAPACK	MPI parallelization library of BLAS and LAPACK functions. The library has about 200 routines.
SSL II/MPI	MPI parallel version of 3D FFT functions

Development Support Tools

- Application development support tools
  - Fujitsu Development Studio provides an integrated development environment, profilers, and a parallel execution debugger as application development support tools.
  - Users of these tools can develop applications efficiently.

Application Development Support Tools	
Integrated development environment	Assistance in development tasks (editing and compiling source code, submitting jobs, checking the job status, retrieving/displaying performance information, etc.) - Eclipse - Parallel Tools Platform (PTP)
Profilers	Assistance in application performance analysis - Instant Performance Profiler - Advanced Performance profiler - CPU performance analysis report
Parallel execution debugger	Assistance in investigating trouble (abnormal end, deadlock, etc.) that occurs during large-scale parallel processing - Abnormal end investigation function - Deadlock investigation function - Debug function using command files

Recommended Compiler Options

- Fortran, C/C++ Trad Mode
  - Performance Focused
  - Precision Focused
- C/C++ Clang Mode

Compile command

- Compile command(not MPI)

Kind	Language	Compile command	Description
Cross compiler	Fortran	frtpx	The command used on the login node.
	C	fccpx	
	C++	FCCpx	
Native compiler	Fortran	frt	The command used on the compute node.
	C	fcc	
	C++	FCC	

- Compile command(MPI)

Kind	Language	Compile command	Description
Cross compiler	Fortran	mpifrtpx	The command used on the login node.
	C	mpifccpx	
	C++	mpiFCCpx	
Native compiler	Fortran	mpifrt	The command used on the compute node.
	C	mpifcc	
	C++	mpiFCC	

Fortran, C/C++ Trad Mode(Performance Focused)

Recommended Options
 (Performance Focused: Fortran, C/C++ Trad Mode)

```
-Kfast,openmp[,parallel]
```

Concept
Specify this option to draw out the full performance of the A64FX. For example, with the option, you can make full use of cores through thread parallelization or SVE through SIMDization, improve instruction-level parallelism by software pipelining, change the operation order by optimization, and use the reciprocal approximation operation.

Compiler Options Induced From -Kfast

- Fortran
  -O3 -Keval,fp_contract,fp_relaxed,fz,ilfunc,mfunc,omitfp,simd_packed_promotion

- C/C++
  -O3 -Keval,fast_matmul,fp_contract,fp_relaxed,fz,ilfunc,mfunc,omitfp,simd_packed_promotion

Compiler Options Induced From -Kfast

Induced Option	Language	Meaning
-O3	Common	Optimization level 3. SIMD and software pipelining operations run at -O2. An additional operation is optimization, such as for unrolling of nested loops.
-Keval	Common	Executes optimization to change how operations are evaluated in a program.
-Kfast_matmul	C/C++	Converts a matrix multiplication loop to a high-speed library call.
-Kfp_contract	Common	Executes optimization for conversion to multiply-add operation instructions.
-Kfp_relaxed	Common	Executes conversion of single- or double-precision floating-point division or the SQRT function to an instruction sequence using a reciprocal approximation operation instruction.
-Kfz	Common	Enables flush-to-zero mode.
-Kilfunc	Common	Inline-expands the intrinsic functions and operations in a program.
-Kmfunc	Common	Converts intrinsic functions and operations to multi-operation functions. This is intended for functions that cannot be used with -Kilfunc.
-Komitfp	Common	Optimizes procedure calls. Consequently, the frame pointer register cannot be guaranteed.
-Ksimd_packed_promotion	Common	Promotes SIMDization with 16-element vectors, assuming that the index computation of array elements made up of single-precision real numbers and also 4-byte integers will not exceed the 4-byte range.

Fortran, C/C++ Trad Mode(Precision Focused)

Recommended Options
 (Precision Focused: Fortran, C/C++ Trad Mode)

```
-Kfast,openmp[,parallel],fp_precision
```

Concept
Use this option when you want to obtain the same precision as -O0 while optimizing performance as much as possible. Specify the new option -Kfp_precision, which suppresses all optimizations that affect precision, as an option appended to the recommended option focused on performance.
This suppresses multiple optimizations that significantly affect performance.

Compiler Options Induced From -Kfp_precision

- Fortran
  -Knoeval,nofp_contract,nofp_relaxed,nofz,noilfunc,nomfunc,parallel_fp_precision

- C/C++
  -Knoeval,nofast_matmul,nofp_contract,nofp_relaxed,nofz,noilfunc,nomfunc,parallel_fp_precision

Compiler Options Induced From -Kfp_precision

Induced Option	Language	Meaning
-Knoeval	Common	Suppresses optimization that changes how operations are evaluated in a program.
-Knofast_matmul	C/C++	Suppresses optimization that converts the loop of matrix multiplication into a high speed library call(matmul).
-Knofp_contract	Common	Suppresses the optimization used for conversion to multiply-add operation instructions.
-Knofp_relaxed	Common	Suppresses the conversion of single- or double-precision floating-point division or the SQRT function to an instruction sequence using a reciprocal approximation operation instruction.
-Knofz	Common	Disables flush-to-zero mode.
-Knoilfunc	Common	Suppresses the inline expansion of the intrinsic functions and operations in a program.
-Knomfunc	Common	Suppresses the conversion of intrinsic functions and operations to multi-operation functions.
-Kparallel_fp_precision	Common	Suppresses optimization that may cause a computation error due to a change in the number of parallel threads.

C/C++ Clang Mode

Recommended Options (for C/C++ Clang Mode)

```
-Nclang -Ofast
```

Concept

Specify this suitable compiler option for OSS compilation.

Compiler Options Induced From -Ofast

- C/C++
  -O3 -ffj-fast-matmul -ffast-math -ffp-contract=fast -ffj-fp-relaxed -ffj-ilfunc -fbuiltin -fomit-frame-pointer -finline-functions

Features of clang mode

- The faster execution performance with modern C code and C++ code is equal to or better than in trad mode.
- The higher usability (GCC compatibility) of OSS application compilation is equal to or better than in trad mode.
- ACLE and FP16 support clang mode only.

Compiler Options Induced From -Ofast

Induced Option	Language	Meaning
-O3	Common	Optimization level 3. The optimization such as loop optimization and inline expansion is performed, as well as more advanced optimization.
-ffj-fast-matmul	C/C++	Same as -Kfast_matmul.
-ffast-math	Common	Same as -Keval.
-ffp-contract=fast	Common	Same as -Kfp_contract.
-ffj-fp-relaxed	Common	Same as -Kfp_relaxed.
-ffj-ilfunc	Common	Same as -Kilfunc.
-fbuiltin	Common	Same as -Klib. Promotes optimization by recognizing the operation of standard library functions
-fomit-frame-pointer	Common	Same as -Komitfp.
-finline-functions	Common	Same as -x-. Specifies to perform Inline expansion of function calls defined in source code.

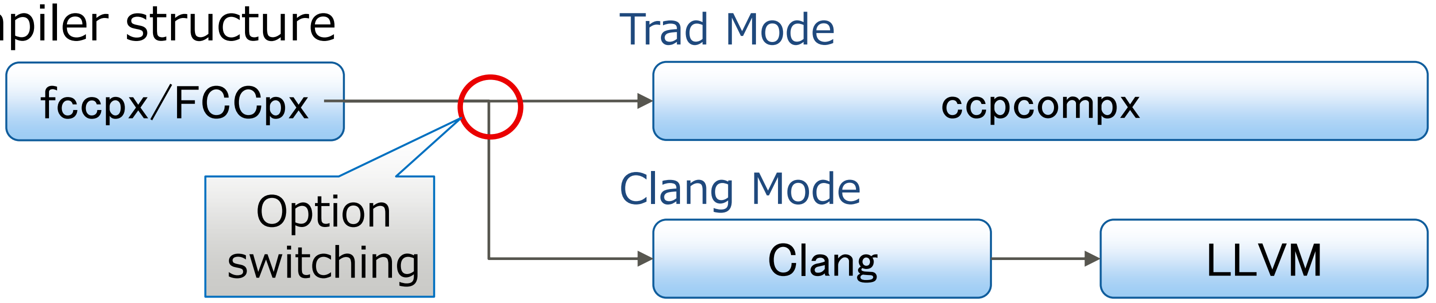
C/C++ Trad Mode and Clang Mode

Assumed Usage Scenarios

- Assumed usage scenarios (use cases)

Usage Scenario	Compiler Suited to Scenario?	
	Trad Mode	Clang Mode
Want to use the resources created on the K computer/FX100 as they are	Yes	No
Want tuning for HPC	Yes	Yes (Options must be specified)
Want to use OSS	No	Yes
Want to use the new language standards C++17	Yes (Partial support)	Yes

- Compiler structure



Overview of Clang Mode External Specifications

- Basic policy as Clang Mode
  - Clang/LLVM view is the base view.
- Mode switching option
  - Use the -Nclang option to switch the compiler.
    Example: FCCpx -Nclang test.cpp
- Functions for compatibility with trad mode
  - Compile information output function (listing function)
    - ➢ The output format (text format) of the current Fujitsu compiler is simple with high readability.
  - Tool output information (loop-related information, etc.)
    - ➢ Because optimization information for loops, etc. is effective for tuning
  - Major optimization options (-K options) and specification-related option (-N options)
    - ➢ Fujitsu's Clang Mode considers portability when compiling K computer/FX100 resources.

      Optimization options:

      -Kfast, -Kilfunc, -Keval, -Kfp_contract, -Kfp_relaxed, etc.

      Specification-related options:

      -Nhook_time, -Nsrc, -Nlst, -Nline, etc.

Comparison of Microarchitecture (Including ISA) Support

FUJITSU

Item	Trad Mode	Clang Mode	Remarks
Supports SVE	Yes	Yes	SIMDization capabilities vary.
Supports fp16	No	Yes	
Specifies variable length for SIMD width	Yes	Yes	Clang Mode currently supports variable lengths only.
Specifies fixed length for SIMD width	Yes	Yes	The default width for Trad Mode is 512 bits. For Clang Mode, see [clang Mode] Specifying Fixed-length SVE SIMD Width
Supports large pages	Yes	Yes	
Supports instruction scheduling latency	Yes	Yes	
Uses zfill	Yes	Yes	
Uses sector cache	Yes	No	
Uses intra-node barrier	Yes	Yes	
Supports prefetch	Yes	Partly	Clang does not support prefetch in some cases.
Utilizes structure instructions	Yes	Yes	

Compiler Function (General, Loop Optimization) Comparison

FUJITSU

	Item	Trad Mode	Clang Mode	Remarks
General	OpenMP	Supported	Supported	Clang Mode supports OpenMP 4.5 (excluding omp declare simd).
	Automatic parallelization	Supported	Not supported	
	ACLE	Not supported	Supported	
	lto	Not supported	Supported	
Loop optimization	Software pipelining	Supported	Partly supported	Specify the option for Clang Mode.
	Loop interchange of nested loops	Supported	Not supported	
	Loop collapse of nested loops	Supported	Not supported	
	Loop unrolling	Supported	Supported	
	Full unrolling	Supported	Supported	
	Loop fission	Partly supported	Partly supported	Specify the option.
	Loop fusion	Supported	Not supported	
	Unswitching	Supported	Supported	
	Multi-versioning	Partly supported	Partly supported	Both support only their individual functions.

Compiler Function (Listing Optimization Information) Comparison

	Item	Trad Mode	Clang Mode	Remarks
Listing optimization display	SIMD	Supported	Supported	
	Unrolling (Number of expansions)	Supported	Supported	
	Full unrolling	Supported	Supported	
	Software pipelining	Supported	Supported	
	Striping/Interleave	Supported	Supported	Trad Mode: STRIPING Clang Mode: INTERLEAVE
	Unswitching	Supported	Supported	
	Loop versioning	Supported	Not supported	
	Clone	Supported	Supported	
	Prefetch (hardware)	Supported	Not supported	
	Prefetch (software)	Supported	Supported	
	Spills	Supported	Supported	
	Fused	Supported	Not supported	
	Collapsed	Supported	Not supported	
	Interchanged	Supported	Not supported	
	Fission	Supported	Supported	
	Multi-operation function	Supported	Not supported	
	Pattern matching (matmul)	Supported	Supported	
	Parallelization information	Supported	Not supported	
Optimization messages		Supported	Partly supported	clang mode outputs messages for SIMD, unrolling, and unswitching.

C/C++ Support for SVE ACLE

- C/C++ Support for SVE ACLE

C/C++ Support for SVE ACLE

- The table below shows C/C++ compiler support for SVE ACLE in relation to modes (Trad/Clang).

✓ : Supported, P: Partly supported, X : Not supported

		SVE	FP16	SVE ACLE (including FP16)
Fujitsu compiler	Trad Mode	✓	X	X
	Clang Mode	✓	✓	✓ *1
Other compilers	Arm compiler (19.2)	✓	✓	✓ *1
	gcc 9.1.0	✓	*2 ✓	X

*1 Support Version 00bet1
https://developer.arm.com/documentation/100987/latest/
(Check the difference from "1.1. 2 change history")
*2 _Float16 for C++ is not supported.

Fixed-length and Variable-length for SVE SIMD Width

- Fixed-length and Variable-length for SVE SIMD Width
- [Clang Mode] Specifying Fixed-length SVE SIMD Width

Fixed-length and Variable-length for SVE SIMD Width

- SVE SIMD width (vector register size)

  Implemented SIMD widths of SVE depend on CPU (A64FX: max. 512 bits).

  The Fujitsu compiler supports options for both fixed- and variable-length SVE SIMD width.

  - Variable-length SIMD width (Trad Mode: -Ksimd_reg_size=agnostic, Clang Mode: -msve-vector-bits=scalable)

    The SVE SIMD width is not considered to be a specific value at compile time, and the executable program decides the SIMD width at execution time. The executable program can be run regardless of SIMD widths implemented in CPUs. This is the default of Clang Mode of C/C++.

  - Fixed-length SIMD width (Trad Mode: -Ksimd_reg_size={128|256|512}, Clang Mode: -msve-vector-bits=512)

    The SVE vector register is considered to be a specific value at compile time. The following optimizations are expected to be promoted. This is the default of Fortran and Trad Mode of C/C++. In Clang Mode of C/C++, the option for fixed-length SVE SIMD width is induced by -ffj-swp and -ffj-zfill.

      ✓ Software pipelining
      ✓ zfill
      ✓ Loop unrolling
      ✓ Loop interleaving
      ✓ Inline expansion of math functions

      (etc.)

[Clang Mode] Specifying Fixed-length SVE SIMD Width (1/2)

FUJITSU

- Examples of programs which expect better performance by specifying fixed-length SIMD width

  Whether better performance can be achieved using fixed-length SIMD width depends on operations in kernel loops. For example, each optimization will be promoted in the following kernel loops.

Kernel loop: loop which represents characteristics of a program

```
    for(i = 0; i < n; i = i+3) {
      y[i] = x1[i] * x2[i];
    }
```

＊ Stride access
Loop interleaving and software pipelining will be promoted

```
    for (i  =  0; i  <  n; i++) {
      b[i] = 0.9 + a[i] * (0.1 + a[i] *
             (0.2 + a[i] * (0.3 + a[i] *
             (0.4 + a[i] * (0.5 + a[i])))));
      }
    }
```

＊ No dependences across iterations
Software pipelining will be promoted

```
    double *x, *y;
    for(i = 0; i < n; i++) {
      x[i] = sin(y[i]);
    }
```

＊ Calling a math function
Inline expansion of a math function and software pipelining will be promoted

[Clang Mode] Specifying Fixed-length SVE SIMD Width (2/2)

- Notes on specifying fixed-length SIMD width

  The following points should be considered when specifying fixed-length SIMD width with Clang Mode.

  - When the program is executed with a SIMD width which is different from one specified at compile time, the result of executions is not guaranteed
  - The -fslp-vectorize option is disabled
  - SIMD built-in functions cannot be used
  - When the -flto option is enabled, fixed-length SIMD width gets disabled
  - For following cases, specifying variable-length SIMD width will generate better-performance code

  ✓ If there are loads/stores of neighboring members of a structure or a complex number in a kernel loop, the “load/store multiple structures” instruction will be exploited for better performance

  ✓ If the number of loop iterations is low, the remainder loop (iterations which run over the SIMD width) is also SIMDized

Compatibility Between Objects Generated by Fujitsu Compiler and Those by Other Compilers

- Compatibility of Objects
- Precautions on Linking

Compatibility of Objects

- Whether objects can be linked
  - The table below shows whether or not objects generated by the Fujitsu compiler can be linked to objects generated by other compilers.
  - If incompatible objects are linked, an error occurs at the link time.

✓: Can be linked,
x : Cannot be linked

Compiler	Language	Fujitsu Compiler (The Fujitsu compiler is used for linking.)			
		Fortran	C * It is not necessary to distinguish between Trad and Clang modes.	C++	
				Trad Mode	Clang Mode
ARM	Fortran	x	✓	x	✓
	C	✓	✓	✓[*1]	✓[*1]
	C++	x	✓	x	✓
LLVM	Fortran	x	✓	x	✓
	C	✓	✓	✓[*1]	✓[*1]
	C++	x	✓	x	✓
GNU	Fortran	x	✓	x	✓
	C	✓	✓	✓[*1]	✓[*1]
	C++	x	✓	x	✓

[*1] Even if objects are generated using the Fujitsu compiler, when linking an object file generated by C++ Clang Mode and an object file generated by C++ Trad Mode, refer to notes in C++ User's Guide 9.6.1.

- Note
  - When using the Profiler or the Debugger for Parallel Applications, use the compile commands of the Fujitsu compiler to compile and link your programs. You cannot use the Profiler or the Debugger for Parallel Applications when using the compile commands of other compilers, such as the GNU compiler.

Precautions on Linking (1/3)

- Precautions on linking with the Arm compiler

		Fujitsu Compiler * The Fujitsu compiler is used for linking.		
	Language	**Fortran**	**C** *** It is not necessary to distinguish between trad and clang modes.**	**C++**
ARM	Fortran	-	Linking not allowed for a program written for another compiler using its own libraries	In both trad and clang modes, linking not allowed for a program written for another compiler using its own libraries
	C	With thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used)	With thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used)	In both trad and clang modes, with thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used)
	C++	-	With thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used)	In clang mode only: - With thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used) - STL (C++ standard library) has to be the same one used by the compiler from another manufacturer. - If "-stdlib=libc++" is specified for compiling by another manufacturer's compiler, "-stdlib=libc++" has to be specified for the Fujitsu compiler too. - If "-stdlib=libstdc++" is specified for compiling by another manufacturer's compiler, "-stdlib=libstdc++" has to be specified for the Fujitsu compiler too.

Precautions on Linking (2/3)

- Precautions on linking with the LLVM compiler

		Fujitsu Compiler * The Fujitsu compiler is used for linking.		
	Language	**Fortran**	**C * It is not necessary to distinguish between trad and clang modes.**	**C++**
LLVM	Fortran	-	Linking not allowed for a program written for another compiler using its own libraries	In both trad and clang modes, linking not allowed for a program written for another compiler using its own libraries
	C	With thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used)	With thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used)	In both trad and clang modes, with thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used)
	C++	-	With thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used)	In clang mode only: - With thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used) - STL (C++ standard library) has to be the same one used by the compiler from another manufacturer. - If "-stdlib=libc++" is specified for compiling by another manufacturer's compiler, "-stdlib=libc++" has to be specified for the Fujitsu compiler too. - If "-stdlib=libstdc++" is specified for compiling by another manufacturer's compiler, "-stdlib=libstdc++" has to be specified for the Fujitsu compiler too.

Precautions on Linking (3/3)

- Precautions on linking with the GNU compiler

		Fujitsu Compiler * The Fujitsu compiler is used for linking.		
	Language	Fortran	C * It is not necessary to distinguish between trad and clang modes.	C++
GNU	Fortran	-	Linking not allowed for a program written for another compiler using its own libraries	In both trad and clang modes, linking not allowed for a program written for another compiler using its own libraries
	C	With thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used)	With thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used)	In both trad and clang modes, with thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used)
	C++	-	With thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used)	In clang mode only: - With thread parallelization, the -Nlibomp option has to be specified for the Fujitsu compiler. (Linking not allowed when the -Nfjomplib option is used) - "-stdlib=libstdc++" has to be specified for the Fujitsu compiler.

Compile Options and Optimization Specifiers Supported in Clang Mode

Compile Option System of Clang Mode

FUJITSU

- Support policy for compile options
  - Support useful options in the -K and -N series for the Fujitsu compiler.
    - Considering the characteristics (fewer loop cycles, etc.) of modern C code and OSS applications, support those options while prioritizing options for optimization of individual instructions and optimization for mathematical functions.
  - Support options in the Clang/LLVM format while prioritizing GCC-compatible options and focusing on compiling and executing OSS applications.

- System supporting compile options
  - -K and -N series of options for the Fujitsu compiler
    - If Clang/LLVM has equivalent options
      - ✓ The options in the -K and -N series are converted to the equivalent options (options in the -f series) for Clang/LLVM.
    - If Clang/LLVM does not have equivalent options
      - ✓ New functions and new options (-ffj series of options) are added to convert the options in the -K and -N series to options in the -ffj series.
      - ✓ Future extension of new functions and new options (-ffj series of options) is planned.
  - Options in the Clang/LLVM format
    - The manual for clang mode describes the range of support.
      - ✓ The manual does not describe unsupported options, which however can still be used.

Optimization Specifier System of Clang Mode

- Support policy for optimization specifiers
  - Support useful optimization specifiers for the Fujitsu compiler in addition to Clang/LLVM optimization specifiers.

- System supporting optimization specifiers
  - Clang/LLVM optimization specifiers
    - The manual for Clang Mode describes the range of support.
      - ✓ The manual does not describe unsupported specifiers, which however can still be used.
  - Fujitsu’s own optimization specifiers
    - Fujitsu’s own optimization specifiers in the #pragma fj format are supported.
      - ✓ The support for the #pragma fj format even in trad mode ensures compatibility between Clang Mode and Trad Mode.
      - ✓ Extension of the range of support is planned, depending on improvements in compatibility with trad mode and on functional extensions.

Options Supported in Clang Mode (1/7)

- Optimization-related options (same as Clang/LLVM)

Option in Clang Mode	Outline	Option in Trad Mode
-Ofast	To create an object program for fast execution on the target machine	-Kfast
-f{builtin|no-builtin}	Whether or not to recognize the actions of standard library functions to promote optimization	-K{lib|nolib}
-f{lto|no-lto}	Whether or not to execute optimization at the link time	-K{lto|nolto}
-f{vectorize|no-vectorize}	Whether or not to generate an object by using the SIMD extension instruction in an intra-loop operation	-K{simd|nosimd}
-f{strict-aliasing| no-strict-aliasing}	Whether or not to execute optimization considering overlapping memory areas in accordance with the strict aliasing rule stipulated in the language standard	-K{strict_aliasing| nostrict_aliasing}
-f{unroll-loops| no-unroll-loops}	Whether or not to execute optimization for loop unrolling	-K{unroll|nounroll}
-f{exceptions| no-exceptions}	Whether or not to define the predefined macro __EXCEPTIONS	-N{exceptions| noexceptions}
-f{omit-frame-pointer| no-omit-frame-pointer}	Whether or not to execute optimization where the frame pointer register is not guaranteed for function calls	-K{omitfp|noomitfp}
-f{inline-functions| no-inline-functions}	Whether or not inline expansion includes the functions defined in the source program	{-x-|-x0}
-f{fast-math|no-fast-math}	Whether or not to execute optimization to change how operations are evaluated	-K{eval|noeval}
-ffp-contract={fast|off}	Whether or not to execute optimization using floating-point multiply-add/subtract operation instructions	-K{fp_contract| nofp_contract}

Options Supported in Clang Mode (/7)

- Optimization-related options (same as Clang/LLVM)

Option in Clang Mode	Outline	Option in Trad Mode
-msve-vector-bits={512|scalable}	Specifies the size of the SVE vector register. (Units are bits.)	(None)
-f{finite-math-only|no-finite-math-only}	To promote the optimization of floating point arithmetic based on the assumption that the argument or operation result is only a finite numerical value.	(None)
-f{reroll-loops|no-reroll-loops}	To apply loop rerolling.	(None)
-f{signed-char|unsigned-char}	To treat char type variable as singed char type.	(None)
-f{slp-vectorize|no-slp-vectorize}	To apply SLP(Superword Level Parallelism) using SIMD instructions.	(None)

Options Supported in Clang Mode (3/7)

- Optimization-related options (Fujitsu-specific, except the prefetch group)

Option in Clang Mode	Outline	Option in Trad Mode
-f{fj-eval-concurrent| fj-no-eval-concurrent}	Whether or not to prioritize instruction parallelism in tree-height-reduction optimization	-K{eval_concurrent| eval_noconcurrent}
-f{fj-fast-matmul| fj-no-fast-matmul}	Whether or not to convert a matrix multiplication loop to a high-speed library call	-K{fast_matmul| nofast_matmul}
-f{fj-fp-relaxed| fj-no-fp-relaxed}	Whether or not to execute reciprocal approximation operation instructions and reciprocal approximation operations using floating-point multiply-add/subtract instructions, for single-precision floating-point division, double-precision floating-point division, and the sqrt function	-K{fp_relaxed|nofp_relaxed}
-f{fj-ilfunc[={loop|procedure}]| fj-no-ilfunc}	Whether or not to inline-expand a math function	-K{ilfunc[={loop|procedure}]| noilfunc}
-f{fj-ocl|fj-no-ocl}	Whether or not to enable optimization control lines specific to the Fujitsu compiler	-K{ocl|noocl}
-f{fj-preex|fj-no-preex}	Whether or not to optimize preceeding evaluation of invariants	-K{preex|nopreex}

Options Supported in Clang Mode (4/7)

- Optimization-related options (Fujitsu-specific, except the prefetch group)

Option in Clang Mode	Outline	Option in Trad Mode
-f{fj-fp-precision| fj-no-fp-precision}	To induce the combination of optimization options that do not cause calculation errors in floating-point operations.	-K{fp_precision| nofp_precision}
-f{fj-hpctag|fj-no-hpctag}	To use the HPC tag address override feature of the A64FX processor.	-K{hpctag|nohpctag}
-f{fj-interleave-loop-insns[=N]| fj-no-interleave-loop-insns}	To apply loop interleaving to SIMD applied loops using SVE. (N is the interleaving count)	(None)
-f{fj-loop-fission| fj-no-loop-fission}	To perform the automatic loop fission optimization for the software pipelining, SIMDization, and the resolution of register shortage.	-K{loop_fission| loop_nofission}
-f{fj-optlib-string| fj-no-optlib-string}	To link the optimized version of the library in the string manipulation function.	-K{optlib_string| nooptlib_string}
-f{fj-swp|fj-no-swp}	To perform software pipelining.	-K{swp|noswp}
-f{fj-zfill[=N]| fj-no-zfill}	To perform zfill optimization.	-K{zfill[=N]|nozfill}

Options Supported in Clang Mode (5/7)

- Optimization-related options (Fujitsu-specific, prefetch group)

Option in Clang Mode	Outline	Option in Trad Mode
-ffj-prefetch-cache-level={1|2|all}	Which cache level to prefetch data to	-Kprefetch_cache_level={1|2|all}
-f{fj-prefetch-conditional| fj-no-prefetch-conditional}	Whether or not to generate prefetch instructions for the array data used in the blocks included in an if statement or switch statement	-K{prefetch_conditional| prefetch_noconditional}
-ffj-prefetch-iteration=**N**	To target the data extracted after **N** cycles of a loop, when generating a prefetch instruction (for the primary cache)	-Kprefetch_iteration=**N**
-ffj-prefetch-iteration-L2=**N**	To target the data extracted after **N** cycles of a loop, when generating a prefetch instruction (for the secondary cache)	-Kprefetch_iteration_L2=**N**
-ffj-prefetch-line=**N**	To target the corresponding data that is prefetched **N** cache lines ahead, when generating a prefetch instruction (for the primary cache)	-Kprefetch_line=**N**
-ffj-prefetch-line-L2=**N**	To target the corresponding data that is prefetched **N** cache lines ahead, when generating a prefetch instruction (for the secondary cache)	-Kprefetch_line_L2=**N**
-f{fj-prefetch-sequential [={auto|soft}]| fj-no-prefetchsequential}	Whether or not to generate prefetch instructions for the array data used in loops and accessed sequentially	-K{prefetch_sequential [={auto|soft}]| prefetch_nosequential}
-f{fj-prefetch-stride| fj-no-prefetch-stride}	Whether or not to generate prefetch instructions for the array data accessed with an even longer stride than the line size of the cache used in a loop	-K{prefetch_stride| prefetch_nostride}
-f{fj-prefetch-strong| fj-no-prefetch-strong}	Whether or not to generate a strong prefetch instruction when generating a prefetch instruction (for the primary cache)	-K{prefetch_strong| prefetch_nostrong}
-f{fj-prefetch-strong-L2| fj-no-prefetch-strong-L2}	Whether or not to generate a strong prefetch instruction when generating a prefetch instruction (for the secondary cache)	-K{prefetch_strong_L2| prefetch_nostrong_L2}
-ffj-no-prefetch	To not generate prefetch instructions	-Knoprefetch

Options Supported in Clang Mode (6/7)

- CPU/Architecture-related, code generation-related, and OpenMP-related options

Option in Clang Mode	Outline	Option in Trad Mode
-mcpu=a64fx	To output the object file for the A64FX processor	-KA64FX
-mcpu=generic	To output the object file for the Arm (universal) processor	-KGENERIC_CPU
-march= arm{v8|v8.1|v8.2|v8.3}-a+sve	To use Armv8.x-A instructions and SVE instructions	-KARCH -KSVE
-march= arm{v8|v8.1|v8.2|v8.3}-a+nosve	To use Armv8.x-A instructions (SVE instructions not used)	-KARCH -KNOSVE
-mcmodel={small|large}	Maximum value for the code area and static data area	-Kcmodel={small|large}
-f{PIC|pic}	To generate position-independent code (PIC)	-K{PIC|pic}
-f{fj-largepage|fj-no-largepage}	Whether or not to use the large page function	-K{largepage|nolargepage}
-f{openmp|no-openmp}	Whether or not to accept directives of OpenMP specifications	-K{openmp|noopenmp}
-f{openmp-simd|no-openmp-simd}	Whether or not to enable only the simd construct and the declare simd construct of OpenMP specifications	-K{openmp_simd|noopenmp_simd}

Options Supported in Clang Mode (7/7)

- Other options

Option in Clang Mode	Outline	Option in Trad Mode
-f{fj-fjcex|fj-no-fjcex}	Whether or not to use the Fujitsu extension functions provided in clang mode	-N{fjcex|nofjcex}
-f{fj-fjprof|fj-no-fjprof}	Whether or not to enable the profiler function	-N{fjprof|nofjprof}
-f{fj-line|fj-no-line}	Whether or not to generate the additional information required for the sampling function during execution as provided by the profiler	-N{line|noline}
-f{fj-hook-time| fj-no-hook-time}	Whether or not to use the hook function with calls from specific locations	-N{hook_time|nohook_time}
-ffj-lst[=p]	To output the source list and statistical information as compiled information to a file	-Nlst[=p]
-ffj-lst=t	To output the source list, statistical information, and more detailed optimization information as compiled information to a file	-Nlst=t
-ffj-lst-out=**file**	To output compiled information to the specified file named **file**	-Nlst_out=**file**
-ffj-src	To output the source list to the standard output	-Nsrc
-Rpass=.*	To output optimization information to the standard error output	-Koptmsg=2
--version	To output the compiler version and copyright information to the standard output	-V

Optimization Specifiers Supported in Clang Mode (1/3)

- Same as Clang/LLVM

Leading "#pragma" omitted

Optimization Specifier in Clang Mode	Outline	Optimization Specifier in Trad Mode
clang fp contract(fast)	To use FMA instructions	(None)
clang fp contract(on)	To use FMA instructions (Equivalent to #pragma STDC FP_CONTRACT ON)	(None)
clang fp contract(off)	To not use FMA instructions	(None)
clang loop unroll(enable)	To unroll loops	loop unroll
clang loop unroll(full)		loop unroll full
clang loop unroll_count(**n**)		loop unroll(**n**)
clang loop unroll(disable)	To not unroll loops	loop nounroll
clang loop vectorize(enable)	To SIMDize loops	loop simd
clang loop vectorize(disable)	To not SIMDize loops	loop nosimd

Optimization Specifiers Supported in Clang Mode (2/3)

- Fujitsu-specific specifiers (except the prefetch group)

Leading "#pragma" omitted

Optimization Specifier in Clang Mode	Outline	Optimization Specifier in Trad Mode
fj loop eval_concurrent	To prioritize instruction parallelism in tree-height-reduction optimization	loop eval_concurrent
fj loop eval_noconcurrent	To suppress instruction parallelism and prioritize use of FMA instructions in tree-height-reduction optimization	loop eval_noconcurrent
fj loop preex	To speculatively evaluate invariants	loop preex
fj loop nopreex	To not speculatively evaluate invariants	loop nopreex

Optimization Specifiers Supported in Clang Mode (3/3)

FUJITSU

- Fujitsu-specific specifiers (prefetch group)

Leading "#pragma" omitted

Clang Mode Optimization Specifier	Outline	Trad Mode Optimization Specifier
fj loop prefetch	To enable the automatic prefetch function of the compiler	loop prefetch
fj loop noprefetch	To disable the automatic prefetch function of the compiler	loop noprefetch
fj loop prefetch_sequential [auto|soft]	To generate prefetch instructions for the array data accessed sequentially	loop prefetch_sequential [auto|soft]
fj loop prefetch_nosequential	To not generate prefetch instructions for the array data accessed sequentially	loop prefetch_nosequential
fj loop prefetch_stride	To generate prefetch instructions for the array data accessed with an even longer stride than the line size of the cache used in a loop	loop prefetch_stride
fj loop prefetch_nostride	To not generate prefetch instructions for stride access	loop prefetch_nostride
fj loop prefetch_strong	To set the prefetch instructions generated for the primary cache as strong prefetch.	loop prefetch_strong
fj loop prefetch_nostrong	To not set the prefetch instructions generated for the primary cache as strong prefetch.	loop prefetch_nostrong
fj loop prefetch_strong_L2	To set the prefetch instructions generated for the secondary cache as strong prefetch.	loop prefetch_strong_L2
fj loop prefetch_nostrong_L2	To not set the prefetch instructions generated for the secondary cache as strong prefetch.	loop prefetch_nostrong_L2

Two OpenMP Libraries

- LLVM OpenMP Library and Fujitsu OpenMP Library
  - LLVM OpenMP Library (-Nlibomp)
  - Fujitsu OpenMP Library (-Nfjomplib)

LLVM OpenMP Library and Fujitsu OpenMP Library (1/2)

- The LLVM OpenMP library is an OpenMP library based on the LLVM OpenMP Runtime Library extended for the A64FX.

OpenMP Library	Option	Supported Functions
LLVM OpenMP library	-Nlibomp (default)	OpenMP 4.5 and Parts of 5.0 Hardware barrier(Default is Software barrier) Sector cache Bind to core (default)
Fujitsu OpenMP library	-Nfjomplib	OpenMP 3.1 Hardware barrier Sector cache Bind to core (Default when execute on job)

- Compiler combination
  - The object files(.o) are common in Fortran and C/C++ Trad Mode, and libraries used can be specified with the –Nlibomp/-Nfjomplib option.
    (If Clang Mode object files are included, only –Nlibomp option is available)

Option	Fortran	C/C++	
		Trad Mode	Clang Mode
-Nlibomp	Available	Available	Available
-Nfjomplib	Available	Available	Not available

LLVM OpenMP Library and Fujitsu OpenMP Library (2/2)

- Selection method
  - Specify in compiler option when linking.
    - -Nlibomp (default) : Use LLVM OpenMP Library
    - -Nfjomplib : Use Fujitsu OpenMP Library
- Difference in specifications
  - Thread stack size

Option	Default size	Environment variables for resizing
-Nlibomp	• 8MiB	OMP_STACKSIZE
-Nfjomplib	• Inherit the process stack size. • If the stack size of the process is specified as unlimited. (Memory size / Number of threads) / 5	OMP_STACKSIZE or THREAD_STACK_SIZE

  - Environment variables
    - The following environment variables are not supported by libomp (be ignored if specified):
      - PARALLEL, FLIB_FASTOMP, THREAD_STACK_SIZE, FLIB_SPINWAIT, FLIB_CPU_AFFINITY, FLIB_NOHARDBARRIER, FLIB_HARDBARRIER_MESSAGE, FLIB_CNTL_BARRIER_ERR, FLIB_PTHREAD, FLIB_CPUBIND, FLIB_USE_ALLCPU, FLIB_USE_CPURESOURCE
  - Shared libraries
    - The following shared libraries will be linked by libomp.
      - libfjomp.so, libfjompcrt.so, libfjomphk.so

LLVM OpenMP Library (-Nlibomp) (1/2)

- By default, the software barrier/sector cache is available.
- To use the hardware barrier, specify the FLIB_BARRIER environment variable.
  - FLIB_BARRIER=HARD : Use the hardware barrier.
  - FLIB_BARRIER=SOFT : Use the software barrier.(default)
- Precautions on using the hardware barrier
  - The number of threads (omp_set_num_threads(), num_threads clause) cannot be controlled.
  - Thread affinity (proc_bind clause, OMP_PLACES, OMP_PROC_BIND) cannot be controlled.
  - Nesting (omp_set_nested(), OMP_NESTED) cannot be controlled.
  - An undeferred task is always generated for a task construct, and the task will not be parallelized.
  - Cancellation is disabled.
- Supports part of OpenMP 5.0 in addition to OpenMP 4.5. Use this library to use the latest OpenMP standard.
- Select the software barrier when using the main functions of OpenMP 4.5/5.0, such as a task or cancellation.

LLVM OpenMP Library (-Nlibomp) (2/2)

- To bind the initial thread to a specific core, use numactl or taskset. This does not affect the upper limit on the number of threads in the program.

```
#!/bin/sh -ex
  :
export FLIB_BARRIER=HARD  # Use the hardware barrier
                           #(Use the software barrier when FLIB_BARRIER is SOFT or not set)
numactl -C12 ./a.out      # Fix the initial thread to C12
```

- For MPI programs, you can use the function of parallelizing memory copy processing in MPI library with threads. Specify -Nlibomp and at least one of the -Kparallel and -Kopenmp as options of a compilation/linkage command.
  - mpifrtpx –Kopenmp –Nlibomp a.f90
- The following environment variables specific to the Fujitsu OpenMP library cannot be used (and will be ignored):
  - PARALLEL
  - FLIB_FASTOMP
  - THREAD_STACK_SIZE
  - FLIB_SPINWAIT
  - FLIB_CPU_AFFINITY
  - FLIB_NOHARDBARRIER
  - FLIB_HARDBARRIER_MESSAGE
  - FLIB_CNTL_BARRIER_ERR
  - FLIB_PTHREAD
  - FLIB_CPUBIND
  - FLIB_USE_ALLCPU
  - FLIB_USE_CPURESOURCE

Fujitsu OpenMP Library (-Nfjomplib)

- Details are omitted since they are the same for existing systems (K computer/FX100).
- By default, the hardware barrier/sector cache is available.
- There is no restriction on generating undeferred tasks while using the hardware barrier.
- OpenMP 3.1 support.
- Specify -Nfjomplib when compiling with the C/C++ compiler in Trad Mode or with the Fortran compiler.
  - Fortran
    - frtpx -Kopenmp -Nfjomplib main.f90
  - C/C++ Trad Mode
    - fccpx -Kopenmp -Nfjomplib main.c
- –Nfjomplib cannot be used with the C/C++ compiler in Clang Mode.
- To bind to core when execute on job. The environment variable FLIB_CPU_AFFINITY can be used to control the bind to core when not execute on job.
- The following environment variable specific to the LLVM OpenMP library cannot be used (and will be ignored):
  - FLIB_BARRIER

Check the Performance

- OpenMP Microbench (Fortran, 12 threads execute)

Directive	[Reference] Skylake (2.7GHz)	libomp		fjomplib	Hardware Barrier Ratio (libomp/fjomplib)	Skylake Ratio (libomp(hard) /Skylake)
		Software Barrier	Hardware Barrier	Hardware Barrier		
	Elapse time (μs)	Elapse time (μs)	Elapse time (μs)	Elapse time (μs)		
PARALLEL	1.10	2.95	2.24	0.43	5.25	2.03
DO/FOR	0.82	1.60	0.13	0.13	0.97	0.16
PARALLEL_DO/FOR	1.12	2.95	2.23	0.45	5.00	2.00
BARRIER	0.76	1.55	0.12	0.12	1.00	0.16
SINGLE	1.18	1.68	1.37	0.60	2.26	1.16
CRITICAL	0.28	0.32	0.32	0.66	0.49	1.14
LOCK/UNLOCK	0.29	0.32	0.32	0.48	0.67	1.12
ORDERED	0.26	0.32	0.32	0.28	1.12	1.22
ATOMIC	0.19	0.65	0.69	0.69	1.00	3.65
REDUCTION	2.25	4.63	2.59	0.95	2.72	1.15

In using libomp library,
Check PARALLEL directives.

Migration From Existing System

- New Options Added in A64FX
- Options With Changed Specifications in A64FX
- Options Deleted From A64FX
- A64FX Compatibility With K Computer/FX10/FX100
  - Fortran compatibility
  - C compatibility
  - C++ compatibility
  - MPI compatibility
  - Math library compatibility
- MPI Extension Functions and Data Types
- Endian

New Options Added in A64FX (1/8)

Compiler Option	Language	Meaning
-Kswp_weak ⭐	Common	Specifies a software pipelining adjustment and a reduction in overlapping executable statements within a loop. Software pipelining is applied to routes at the execution time, and the required number of loop iterations passing through the routes is small. For this reason, performance can be expected to improve in cases where the number of loop iterations is small but not known at the compile time.
-Kswp_freg_rate=N ⭐ -Kswp_ireg_rate=N ⭐ -Kswp_preg_rate=N	Common	Changes the application conditions regarding the registers for software pipelining. N is an integer value between 1 and 1,000, specifying the percentage of registers available for software pipelining. When the specified value is larger than 100, an increase in the number of actual registers may be determined, and software pipelining may be applied with them. However, execution performance may decrease due to insufficient registers since this shortage increases the number of instructions saved to memory and restored. - freg: Floating-point register and SVE vector register - ireg: Integer register - preg: SVE predicate register
-Kloop_fission_threashold=N ⭐	Common	Specifies the threshold value that determines the loop granularity (number of instructions or registers in a loop, etc.) after loop fission. N is a value ranging from 1 to 100. The default is 50. The smaller the value for N, the smaller the loops after fission and the greater the number of splits.
-Kloop_fission_ stripmining[={N|L1|L2}] -Kloop_fission_nostripmining	Common	-Kloop_fission_stripmining[={N|L1|L2}] Specifies strip mining optimization during loop fission. N is a value ranging from 1 to 100,000,000, specifying the strip length. If the characters L1 or L2 are specified, the specified cache is adjusted to match the expected length. If neither is specified, the compiler automatically makes a determination.

⭐ Option you should take note of

New Options Added in A64FX (2/8)

Compiler Option	Language	Meaning
-Kassume ★ ={shortloop| memory_bandwidth| time_consuming_compilation}	Common	Specifies an option to work at -O1 or higher. -Kassume=shortloop Controls optimization by assuming a low number of cycles when the number of innermost loop cycles in a program is not known. -Kassume=memory_bandwidth Controls optimization by assuming the innermost loop in a program is a bottleneck due to memory bandwidth. -Kassume=time_consuming_compilation Controls optimization to shorten the compile time. (As with -O level control, rather than specific functions being disabled, the optimization is more restricted/suppressed as the program grows larger.)
-Keval_[no]concurrent ★	Common	-Keval_concurrent Makes a transformation to prioritize operation instruction parallelism in tree-height-reduction optimization. The option is specified to work at -O1 or higher together with -Keval. Performance is likely to improve when this option is applied to loops having many operations that cannot be software pipelined. By default, -Keval_noconcurrent considers linkage with software pipelining, suppresses instruction parallelism, and makes a transformation to prioritize the use of FMA instructions.
-Kloop_[no]perfect_nest	Common	-Kloop_noperfect_nest Specifies suppression of optimization that fissions imperfectly nested loops into perfectly nested loops. This option turns currently running functions into optional functions. The default at -O2 or lower is -Kloop_noperfect_nest. The default at -O3 is -Kloop_perfect_nest.

New Options Added in A64FX (3/8)

Compiler Option	Language	Meaning
-K[no]optlib_string ⭐	Common	-Koptlib_string Links the optimized library of character string manipulation functions. The default is -Knooptlib_string.
-K[no]preload ⭐	Common	-Kpreload Runs optimization aimed at speeding up executable modules by speculatively executing a load instruction before the if condition is determined from the then/else clause of an if statement. The default is -Knopreload.
-K[no]sibling_calls	Common	-Ksibling_calls Optimizes trailing calls. The default actions are at -O2 or higher.
-Ksimd_reg_size ={128|256|512|agnostic}	Common	Generates instructions with an assumption that the specified value is the SVE vector register size. If **agnostic** is specified, executable instructions are generated irrespective of the size of the SVE vector register of the CPU.
-Ksimd_[no]use_multiple _structures	Common	-Ksimd_nouse_multiple_structures Specifies not to use the load/store multiple-structures instruction of SVE. This option is available only when -Ksimd and -KSVE are enabled. The default is -Ksimd_use_multiple_structures.
-Ksimd_[no]uncounted_loop	Common	-Ksimd_uncounted_loop Implementation of SIMD "do while" loops, "do until" loops, and "do" loops containing a loop ending statement (only a limited range is optimized). The default is -Ksimd_nouncounted_loop.

New Options Added in A64FX (4/8)

Compiler Option	Language	Meaning
-K[no]sch_pre_ra ☆	Common	-Knosch_pre_ra Suppresses register preallocation instruction scheduling. -Ksch_pre_ra is applied at –O1 or higher. Performance is likely to improve when this option is used where many spills occur.
-K[no]sch_post_ra	Common	-Knosch_post_ra Suppresses register post-allocation instruction scheduling. -Ksch_post_ra is applied at -O1 or higher.
-Kunroll_and_jam[=N] -Knounroll_and_jam	Common	-Kunroll_and_jam[=N] Optimizes unrolling and jamming (outer loop unrolling). N is a value ranging from 2 to 1,000, setting an upper limit on the number of loop expansions. An option is specified to work at -O2 or higher.
-K{align_loops[=N] |noalign_loops}	Common	-Kalign_loops[=N] Aligns the top alignment of loops to an N-byte boundary, where is N is a power of 2. N is a value ranging from 0 to 32,768 (powers of 2), specifying a byte boundary for the top alignment of loops. If N is omitted or N=0 is specified, the compiler automatically makes a determination. The default actions are at -O2 or higher.
-Kopenmp_[no]collapse _except_innermost	Common	-Kopenmp_collapse_except_innermost Excludes the innermost loop from collapse processing. This may prevent collapse processing from causing execution performance to deteriorate. This option is enabled when -Kopenmp is specified. The default is -Kopenmp_nocollapse_except_innermost.
-K[no]openmp_simd	Common	-Kopenmp_simd Enables only the SIMD construct and DECLARE SIMD construct of OpenMP specifications.
-Kcmodel={small|large}	Common	Specifies the memory model. The default is small.

New Options Added in A64FX (5/8)

Compiler Option	Language	Meaning
-K[no]pc_relative_ literal_loads	Common	-Kpc_relative_literal_loads Handles the code area within a procedure as 1 MB or less, and accesses a literal pool with a single instruction. The default is -Knopc_relative_literal_loads.
-K[no]plt	Common	-Kplt Specifies whether to use the procedure linkage table (PLT) to access the external symbols in position-independent code (PIC). This option has meaning only together with -K{pic|PIC}. The default is -Kplt.
-Ktls_size={12|24|32|48} ★	Common	Specifies the required offset size for access to thread-local storage. The unit is bits.
-KARM{V8_A|V8_1_A| V8_2_A|V8_3_A}	Common	Generates an object file containing the specified instruction set. The default is -KARMV8_3_A.
-K[NO]SVE	Common	Specifies whether to use SVE extension instructions. The default is -KSVE.
-KA64FX	Common	Specifies that objects be generated for the A64FX processor (default).
.-KGENERIC_CPU	Common	Specifies that objects be generated for the universal ARM processor.
-K[no]hpctag	Common	-Khpctag Specifies the use of the HPC tag address override function of the A64FX processor. The HPC tag address override function enables the sector cache function and hardware prefetch assist function (hardware prefetch function for stride access, etc.). For the Fugaku supercomputer, this is enabled by default. -Knohpctag is used to batch suppress the above-described functions.

New Options Added in A64FX (6/8)

Compiler Option	Language	Meaning
-K[no]array_declaration_opt	Common	Specifies to perform the optimization such as SIMDization assuming that the array subscript does not exceed the range of the array declaration. The default actions are at -O1 or higher.
-K[no]extract_stride_store	Common	Specifies to use scalar instructions for stride access store in the target loop when using SIMD extensions. The default is -Knoextract_stride_store.
-K[no]fp_precision ☆	Common	Specifies to induce the combination of optimization options that do not cause calculation errors in floating-point operations. The default is -Knofp_precision.
-K[no]subscript_opt	F	These options specify whether or not to perform optimizations to prioritize neighboring data regarding array subscript, which is applicable to stencil calculation. The default is -Knosubscript_opt.
-Kswp_policy= { auto | small | large } ☆	Common	Specifies a policy to select an instruction scheduling algorithm used in software pipelining. The default is -Kswp_policy=auto. swp_policy=auto The compiler automatically selects a fit algorithm for each loop. swp_policy=small An algorithm fit for a small loop, such as a loop with low register pressure, is used. swp_policy=large An algorithm fit for a large loop, such as a loop with high register pressure, is used.

New Options Added in A64FX (7/8)

Compiler Option	Language	Meaning
-X08	F	Fortran 2008 language specification level option. This is induced in compilation of a.f08, a.F08, a.for, and a.FOR.
-std={c11|gnu11}	C	Compiles with GNU11. (default)
-std={c++17|gnu++17}	C++	Compiles with GNU++17.
-N[no]clang ⭐	C/C++	Runs the Clang/LLVM-based compiler (Clang Mode).
-N[no]coverage --[no-]coverage	Common	-Ncoverage / --coverage Generates information to use the code coverage function. The default is -Nnocoverage / --no-coverage.
-Nnocheck_global	F	-Nnocheck_global Adds the negative option of -Ncheck_global. -Ncheck_global is induced from -Nquickdbg and -Eg, without compile error detection, to support cases that require error detection only at the execution time. This is specified as -Nquickdbg -Nnocheck_global.
-Nfmtl ={serial|SSL2|parallel|SSL2BLAMP}	C++	Specifies the use of the Fujitsu matrix template library. - serial: Sequential version - SSL2: High-speed sequential version using the SSL2 library - parallel: Thread parallel version - SSL2BLAMP: High-speed thread parallel version using the SSL2 library

New Options Added in A64FX (8/8)

Compiler Option	Language	Meaning
-Nfjomplib ☆	Common	Uses the OpenMP library specific to Fujitsu as a library used for parallel processing.
-Nlibomp ☆	Common	Uses the LLVM OpenMP library as a library used for parallel processing. In clang mode, -Nlibomp is always specified.
-N[no]reordered_variable_stack	Common	-Nreordered_variable_stack Allocates automatic variables to the stack area in ascending order of data size. The default is -Nnoreorederd_variable_stack, which allocates variables in the order of program declaration statements.

Options With Changed Specifications in A64FX (1/7)

Compiler Option	Language	Meaning Reason for Change	Old Specification	New Specification
-Kfast	Common	Creates an object program for fast execution on the target machine. To pursue universal performance improvements. -Kprefetch_conditional may cause performance deterioration and was thus deleted. This change was made along with specification changes of other functions.	Fortran -O3 -K**dalign**,eval, fp_contract,fp_relaxed, ilfunc,mfunc,**ns**,omitfp,**prefet ch_conditional** C/C++ -O3 -K**dalign**,eval, fast_matmul,fp_contract, fp_relaxed,ilfunc,lib,mfunc,**ns**, omitfp,**prefetch_conditional**, rdconv -x-	Fortran -O3 -Keval,fp_contract, fp_relaxed,**fz**,ilfunc,mfunc,omit fp,**simd_packed_promotion** C/C++ -O3 -Keval,fast_matmul, fp_contract,fp_relaxed, **fz**,ilfunc,mfunc,omitfp, **simd_packed_promotion**
-Kauto	F	Allocates the local variables, excluding those with the SAVE attribute or an initial value, on the stack. To speed up with the default	Individually specified to work	Works by default from -O0
-Ktemparraystack	F	Allocates the interim results of array operations and the evaluation results of mask expressions on the stack. To speed up with the default	Individually specified to work	Works by default from -O0
-Kautoobjstack	F	Allocates automatically allocated data objects on the stack. To speed up with the default	Individually specified to work	Works by default from -O0

Options With Changed Specifications in A64FX (2/7)

Compiler Option	Language	Meaning Reason for Change	Old Specification	New Specification
-Klib	C/C++	Recognizes sin, etc. as standard library functions rather than user-defined functions. To adapt to industry standards	Enabled at -O1 or higher Induced from –Kfast	⭐ Enabled at -O0 or higher Works by default from -O1
-x-	C/C++	Performs inline expansion. Changed according to the defaulting to –Xg	Induced from -Kfast	Induced from -O3
-x=quick	C++	Uses inline to prevent enormous functions. For the expected positive effects generally	-x=noquick (default)	-x=quick (default)
-Xg	C/C++	Specifies compiling based on the language specification of GNU C/C++ compilers. To accept the extended specifications of GNU C/C++ compilers by default	3 language specification modes: -Xa, -Xc, -Xg Default is -Xa, and others are optional * Announced obsolescence of old specifications, compile continuing in GNU C/C++-compatible mode	⭐ Deletes -Xa and -Xc, and defaults to -Xg
-Klto(F) -flto(C/C++) ⭐	Common	LTO (link time optimization) • ipo and lto integration with lto • LTO behavior in Fortran	-Kipo (C only) (Note) Warning about old specifications, compile continuing	-Klto (F) -flto (C/C++ Clang Mode)

Options With Changed Specifications in A64FX (3/7)

Compiler Option	Language	Meaning Reason for Change	Old Specification	New Specification
-Kzfill ★	Common	Speeds up store instructions. This is very similar to prefetch of the store area except that data is not loaded from memory and only a cache location is allocated. Changed based on architecture specifications	-KXFILL ("X" means fill, which is indeterminate value) * Warning about old specifications, compile continuing	-Kzfill ("z" means zero fill)
-Kilfunc [={loop|procedure}]	Common	Inline-expands intrinsic functions and operations. For the expected positive effects in various applications	Default for -Kilfunc is -Kilfunc=loop	Default for -Kilfunc is -Kilfunc=procedure
-Kprefetch_stride ★ [={soft|hard_auto|hard_always}]	Common	Generates a prefetch instruction for the array data accessed with an even longer stride than the cache line size in a loop. To add a function making use of the microarchitecture	-Kprefetch_stride	Enables use of hardware stride prefetch function -Kprefetch_stride [={soft|hard_auto|hard_always}]
-Kprefetch_strong[_L2]	Common	Specifies that L1/L2 prefetch be a strong prefetch instruction. Specifications changed based on the microarchitecture	Default is -Kprefetch_nostrong	Default is -Kprefetch_strong

Options With Changed Specifications in A64FX (4/7)

Compiler Option	Language	Meaning Reason for Change	Old Specification	New Specification
-Kassume=shortloop	Common	Specifies optimization control assuming a low number of innermost loop cycles in a program. To integrate into the optimization control option based on application features	-Kshortloop (Note) Warning about old specifications, compile continuing	-Kassume=shortloop
-Kloop_part_simd	Common	Implementation of SIMD parts of a loop by splitting the parts that can be SIMDized from the parts that cannot. To have part of the SIMD function rather than a split	-Kloop_part_simd works as part of -Kloop_fission function	Works when -Ksimd is enabled
-Kloop_part_parallel	Common	Parallelizes the threads in parts of a loop by splitting the parts that can be thread-parallelized from the parts that cannot. To have part of the parallelization function rather than a split	-Kloop_part_parallel works as part of -Kloop_fission function	Works when -Kparallel is enabled
-Koptions	F	Treats lines that begin with the !options line as compiler directive lines. Though deletion was initially considered, it was decided to keep only -O[1-3] (without OCL).	!options -O[1-3] Compiler option with part of !options	!options -O[1-3]

Options With Changed Specifications in A64FX (5/7)

Compiler Option	Language	Meaning Reason for Change	Old Specification	New Specification
-Krdconv[={1|2}]	C/C++	Executes optimization assuming signed integer-type expressions of 4 bytes or less do not overflow and unsigned integer-type expressions of 4 bytes or less do not wrap around. To break up the -Krdconv function and provide the default option as an industry standard specification	-Krdconv Promotes optimization assuming 4-byte signed variables do not overflow	-Krdconv derives -Krdconv=1. Derivations from -Kfast deleted since -Krdconv is enabled at -O2 -Krdconv=1 Promotes optimization assuming 4-byte signed integer-type expressions do not overflow -Krdconv=2 Promotes optimization assuming 4-byte unsigned integer-type expressions do not wrap around
-Kstrict_aliasing ☆	C/C++	Specifies optimization considering overlapping memory areas in accordance with the strict aliasing rule. To change to the industry standard level of specifications (including option names)	-Kmemalias ＊ Recommendation against, planned obsolescence of, and warning about old specifications, -Krestrict_aliasing announcement, compile continuing	-Kstrict_aliasing
-Kfz	Common	Specifies the use of flush-to-zero mode. Changed based on architecture specifications	-Kns (NS bit for SPARC) ＊ Obsolescence announcement, compile continuing	-Kfz (FZ bit for ARM)

Options With Changed Specifications in A64FX (6/7)

Compiler Option	Language	Meaning Reason for Change	Old Specification	New Specification
-Kalign_commons	F	Specifies adjustment of the 8-byte boundary on 8-byte integer-type, double/quadruple-precision, real/complex-number-type data in allocation processing of variables belonging to common blocks to storage space. To split older functions and change to only those specifications required considering industry standards	-Kdalign (Default is -Knodalign) -Kdalign has 2 functions: (1) Function to align members of common variables. (2) Function in SPARC to specify whether to use ldd instruction -> Not required ＊ Warning about old specifications, compile continuing	Fortran -Kalign_commons(default) C/C++ Deleted
-Nline	C/C++	Generates the additional information required for the sampling function during execution as provided by the profiler. To prioritize user convenience on the K computer/FX100	-Nline (-Xa(default), together with -Xc) -Nnoline (together with -Xg)	Default is -Nline when -Xg becomes default
-std={c++14|gnu++14}	C++	Compiles with GNU++14. Specifications changed according to full support for C++14	C++14 optional	Default is C++14

Options With Changed Specifications in A64FX (7/7)

Compiler Option	Language	Meaning Reason for Change	Old Specification	New Specification
-gdwarf[-4]	Common	Supports DWARF version 4. Changed according to DWARF4 support	-gdwarf-2 (DWARF version 2) ＊ Obsolescence announcement for old specifications, compile continuing	-gdwarf[-4] (DWARF version 4)

-Ncheck_std=
{03d|03e|03o|03s}
 	
-Ncheck_std=
|08d|08e|08o|08s}
	F	Executes the language standard test on the source program.
		• Option name change preferred since it may have been confusing due to the use of the -v option for different features by other manufacturers
		• Support of the Fortran 2008 standard
			-v{03d|03e|03o|03s}
			＊ Warning about old specifications, compile continuing
				-Ncheck_std=
				{03d|03e|03o|03s|08d|
				08e|08o|08s}

Options Deleted From A64FX (1/2)

Compiler Option	Language	Reason for Deletion	Behavior
-KHPC_ACE[2]	Common	Because the option is used in the K computer/FX100 SIMD extension instruction set	Warns, and continues compile
-K[NO]FLTLD	Common	Because the option is intended for the K computer/FX100 architecture	Warns, and continues compile
-K[NO]GREG_APPLI	C/C++	Because the option is for intended for the SPARC-series architecture	Warns, and continues compile
-Kadr{44|64}	Common	Because the option is intended for the SPARC architecture	Warns, and continues compile
-K[no]fed	Common	Because the option is intended for the FX100 architecture	Warns, and continues compile
-Kfuncalign=N	C	Because the option is intended for the K computer/FX100 architecture	Warns, and continues compile
-Kloop_[no]fission_if	Common	This function fissions a loop containing an IF statement but had few users and may have had a risk of performance decrease. Since it was replaced by a new loop fission function that fissions a loop after branching by SIMDization, the option was deleted.	Warns, and continues compile
-K[no]nf	Common	Because the option is intended for the FX100 architecture	Warns, and continues compile
-Kopenmp_tls	Common	Because the threadprivate implementation of OpenMP was changed from a Fujitsu-specific method to TLS only, which is the industry standard	Announces deletion, and continues compile
-Ksimd_[no]separate_stride	Common	Because the option is in the stride SIMD instruction for the FX100.	Warns, and continues compile
-K[no]uxsimd	Common	Deleted because the option served as a 2-element vector SIMD function for the K computer/FX10	Warns, and continues compile
-Kvppocl	F	Because the option is a compatibility option for conventional vector machines no longer used by anyone	Announces deletion, and continues compile

Options Deleted From A64FX (2/2)

Compiler Option	Language	Reason for Deletion	Behavior
-Nrt_tune_io	C/C++	The Fugaku supercomputer has a performance analysis tool that outputs the input/output information that was output as a runtime output function.	Warns, and continues compile
-Nstl={500|521|500fast} -Kstl_fast_new	C++	Because STLport-series STL was deleted and libc++ was changed to the default STL	Warns, and continues compile
-fcall_used_g7 -ffixed-g{4|5}	Common	Because the option is intended for the SPARC architecture	Warns, and continues compile
-fvisibility={default|internal|hidden|protected}	C/C++	Eliminated in Trad Mode only, because it can be substituted by the same name option in Clang Mode.	Warns, and continues compile

A64FX Compatibility With K Computer/FX10/FX100 (1/11)

- Fortran compatibility (1/3)
  - For details, ask your system administrator.

No.	Target Machine	Item
1	[K computer/FX10/FX100]	Defaulting to the -Kauto option
2	[K computer/FX10/FX100]	Defaulting to the -Ktemparraystack option
3	[K computer/FX10/FX100]	Defaulting to the -Kautoobjstack option
4	[FX100]	Change in the handling of options when the -Klto option is specified together with the -S or -Wa option
5	[K computer/FX10]	Changed default value for the level in the -Ksimd[=level] option
6	[K computer/FX10/FX100]	Results of intrinsic functions of the GAMMA type
7	[K computer/FX10/FX100]	Output results of G-format editing
8	[K computer/FX10/FX100]	Return value for the IOSTAT specifier at the execution time of a nonadvancing output statement
9	[K computer/FX10/FX100]	Changed specifications of the runtime information output function
10	[K computer/FX10/FX100]	LLVM OpenMP library installation
11	[K computer/FX10/FX100]	Unsupported items in Fortran
12	[K computer/FX10/FX100]	Changed values for predefined version macros
13	[K computer/FX10/FX100]	Object compatibility when using OpenMP
14	[K computer/FX10/FX100]	Compile error testing compliant with Fortran standards
15	[K computer/FX10/FX100]	New/Changed specifications of Fortran standards, and kind type parameters for real numbers and complex numbers (extended specifications)
16	[K computer/FX10/FX100]	Compile error testing according to co-arrays and OpenMP specifications
17	[K computer/FX10/FX100]	Message output at the execution time of a program having an error when the compile option -H is enabled
18	[K computer/FX10/FX100]	Change from -X03 to -X08 in the language specification level options induced from the file suffixes .F, .f, .FOR, and .for
19	[K computer/FX10/FX100]	Runtime message output in accordance with Fortran standards
20	[K computer/FX10/FX100]	Change of the -v03d, -v03e, -v03o, and -v03s options to -Ncheck_std=
21	[K computer/FX10/FX100]	Changes in values for macros and named constants along with support for the OpenMP API version 4.0 specifications
22	[K computer/FX10/FX100]	Deleted compile option -K{openmp_tls|openmp_notls}

A64FX Compatibility With K Computer/FX10/FX100 (2/11)

- Fortran compatibility (2/3)
  - For details, ask your system administrator.

No.	Target Machine	Item
23	[K computer/FX10/FX100]	Error testing for the OpenMP THREADPRIVATE directive
24	[K computer/FX10/FX100]	Deleted options specific to the SPARC instruction set architecture
25	[K computer/FX10/FX100]	Deleted compile options specific to HPC-ACE and HPC-ACE2
26	[FX100]	Changed specifications dependent on the instruction set architecture
27	[K computer/FX10/FX100]	Deleted -Kvppocl option
28	[K computer/FX10/FX100]	Compile error testing for OpenMP directives
29	[K computer/FX10/FX100]	Deleted compiler command line "!options" and -Koptions option
30	[K computer/FX10/FX100]	Deleted -Nuse_rodata option, and changed inductive relationship
31	[K computer/FX10/FX100]	Deleted -Q option
32	[K computer/FX10/FX100]	Changed -K{XFILL|NOXFILL} option, optimization specifier {XFILL|NOXFILL}, and optimization information output
33	[K computer/FX10/FX100]	Changed specifications of traceback of procedures containing an ENTRY statement
34	[K computer/FX10/FX100]	Deleted -K{shortloop=N|noshortloop} option and optimization specifier {SHORTLOOP(n)|NOSHORTLOOP}
35	[K computer/FX10/FX100]	Deleted -K{uxsimd|nouxsimd} option and optimization specifier {UXSIMD|NOUXSIMD}
36	[K computer/FX10/FX100]	Changed specification for the -Kswp and -Kswp_strong options
37	[K computer/FX10/FX100]	Changed display format of compiled information (SIMDization)
38	[K computer/FX10/FX100]	Changed display format of compiler messages (loop fission)
39	[K computer/FX10/FX100]	Changed display format of compiled information (loop fission)
40	[K computer/FX10/FX100]	Changed display format of compiled information (software pipelining)
41	[K computer/FX10/FX100]	Deleted -Kdalign option
42	[K computer]	Changed deduction relationship between the optimization level and the compile option -Kloop_fission
43	[K computer/FX10/FX100]	Changed compile option -Kloop_part_simd and optimization specifier LOOP_PART_SIMD
44	[K computer/FX10/FX100]	Changed compile option -Kloop_part_parallel and optimization specifier LOOP_PART_PARALLEL

A64FX Compatibility With K Computer/FX10/FX100 (3/11)

- Fortran compatibility (3/3)
  - For details, ask your system administrator.

No.	Target Machine	Item
45	[K computer/FX10/FX100]	Deleted compile option -K{loop_fission_if|loop_nofission_if}
46	[K computer/FX10/FX100]	Change of -Kprefetch_strong to default
47	[K computer/FX10/FX100]	Requirement that -Khpctag be enabled for -Kprefetch_nostrong and -Kprefetch_nostrong_L2
48	[K computer]	Deleted compile options specific to the K computer
49	[FX10]	Deleted compile options specific to the FX10
50	[FX100]	Defaulting to the SIMDization function through multiple redundant executions with the SIMD length, and deleted optimization specifier simd_redundant_vl
51	[K computer/FX10/FX100]	Modified runtime messages
52	[K computer/FX10/FX100]	Changed behavior when a specified option is unrecognizable
53	[K computer/FX10/FX100]	Changed options induced from the compile option -Kfast
54	[K computer/FX10/FX100]	Changed execution results of formatted sequential access input/output and formatted nonadvancing input/output
55	[K computer/FX10/FX100]	Runtime messages that are output during parallel processing
56	[FX10/FX100]	Type conversion option in the LLVM OpenMP library
57	[K computer/FX10/FX100]	IO buffer parallel transmission with the LLVM OpenMP library
58	[K computer/FX10/FX100]	Runtime messages that are output when capturing floating-point exceptions
59	[K computer/FX10/FX100]	Changed Japanese message for jwd8205o-i
60	[K computer/FX10/FX100]	Changed Fortran 2008 specifications for the intrinsic procedures FRACTION, RRSPACING, and SPACING

A64FX Compatibility With K Computer/FX10/FX100 (4/11)

- C compatibility (1/3)
  - For details, ask your system administrator.

No.	Target Machine	Item
1	[K computer/FX10/FX100]	Defaulting to the -Klib option
2	[K computer/FX10/FX100]	Deleted -Kipo option
3	[FX100]	Change in the handling of options when the -Klto option is specified together with the -S or -Wa option
4	[K computer/FX10]	Changed default value for the level in the -Ksimd[=level] option
5	[K computer/FX10/FX100]	Changed specifications of the runtime information output function
6	[K computer/FX10/FX100]	LLVM OpenMP library installation
7	[K computer/FX10/FX100]	Deleted -gdwarf-2 option, and added -gdwarf and -gdwarf-4 options
8	[K computer/FX10/FX100]	Deleted compile option -noansi
9	[K computer/FX10/FX100]	Changed default value for the compile option -f{signed-char|unsigned-char}
10	[K computer/FX10/FX100]	Changed values for predefined version macros
11	[K computer/FX10/FX100]	Deleted compile option -X (The language specification mode is GNU-compatible mode only.)
12	[K computer/FX10/FX100]	Changed GNU C compatible version
13	[K computer/FX10/FX100]	Changed default value for the compile option -std
14	[K computer/FX10/FX100]	Changed definitions of the macros __STRICT_ANSI__, linux, unix, and __STDC_VERSION__
15	[K computer/FX10/FX100]	Change of the language specification level when the compile option -ansi or -std is enabled
16	[K computer/FX10/FX100]	Changed behavior when the compile options -Dname and -Uname are specified together
17	[K computer/FX10/FX100]	Changed behavior when the compile option -P is specified
18	[K computer/FX10/FX100]	Changed default value for the compile option -N{line|noline}
19	[K computer/FX10/FX100]	Object compatibility when using OpenMP
20	[K computer/FX10/FX100]	Deleted compile option -K{openmp_tls|openmp_notls}
21	[K computer/FX10/FX100]	Deleted options specific to the SPARC instruction set architecture
22	[K computer/FX10/FX100]	Deleted GNU C compatible options specific to the SPARC instruction set architecture

A64FX Compatibility With K Computer/FX10/FX100 (5/11)

FUJITSU

- C compatibility (2/3)
  - For details, ask your system administrator.

No.	Target Machine	Item
23	[K computer/FX10/FX100]	Deleted compile options specific to HPC-ACE and HPC-ACE2
24	[FX100]	Changed specifications dependent on the instruction set architecture
25	[K computer/FX10/FX100]	Changed specifications of options related to -Kstrict_aliasing
26	[K computer/FX10/FX100]	Deleted -Kfuncalign=N option
27	[K computer/FX10/FX100]	Deleted -Nuse_rodata option, and changed inductive relationship
28	[K computer/FX10/FX100]	Changed options induced from the compiler options -O2 and -O3
29	[K computer/FX10/FX100]	Changed -K{XFILL|NOXFILL} option, optimization specifier {XFILL|NOXFILL}, and optimization information output
30	[K computer/FX10/FX100]	Deleted -K{shortloop=N|noshortloop} option and optimization specifier {SHORTLOOP(n)|NOSHORTLOOP}
31	[K computer/FX10/FX100]	Deleted -K{uxsimd|nouxsimd} option and optimization specifier {UXSIMD|NOUXSIMD}
32	[K computer/FX10/FX100]	Changed specifications of the -Kswp and -Kswp_strong options
33	[K computer/FX10/FX100]	Changed display format of compiled information (SIMDization)
34	[K computer/FX10/FX100]	Changed display format of compiler messages (loop fission)
35	[K computer/FX10/FX100]	Changed display format of compiled information (loop fission)
36	[K computer/FX10/FX100]	Changed display format of compiled information (software pipelining)
37	[K computer/FX10/FX100]	Deleted -Kdalign option
38	[K computer/FX10/FX100]	Changed specifications of the built-in functions __sync_fetch_and_nand and __sync_nand_and_fetch
39	[K computer]	Changed deduction relationship between the optimization level and the compile option -Kloop_fission
40	[K computer/FX10/FX100]	Changed compile option -Kloop_part_simd and optimization specifier LOOP_PART_SIMD
41	[K computer/FX10/FX100]	Changed compile option -Kloop_part_parallel and optimization specifier LOOP_PART_PARALLEL
42	[K computer/FX10/FX100]	Deleted compile option -K{loop_fission_if|loop_nofission_if}
43	[K computer/FX10/FX100]	Change of -Kprefetch_strong to default
44	[K computer/FX10/FX100]	Requirement that -Khpctag be enabled for -Kprefetch_nostrong and -Kprefetch_nostrong_L2

A64FX Compatibility With K Computer/FX10/FX100 (6/11)

- C compatibility (3/3)
  - For details, ask your system administrator.

No.	Target Machine	Item
45	[K computer]	Deleted compile options specific to the K computer
46	[FX10]	Deleted compile options specific to the FX10
47	[FX100]	Defaulting to the SIMDization function through multiple redundant executions with the SIMD length, and deleted optimization specifier simd_redundant_vl
48	[K computer/FX10/FX100]	Modified runtime messages
49	[K computer/FX10/FX100]	Changed behavior when a specified option is unrecognizable
50	[K computer/FX10/FX100]	Changed options induced from the compile option -Kfast
51	[K computer/FX10/FX100]	Runtime messages that are output during parallel processing
52	[K computer/FX10/FX100]	Runtime messages that are output when capturing floating-point exceptions
53	[K computer/FX10/FX100]	Changed Japanese message for jwd8205o-i

A64FX Compatibility With K Computer/FX10/FX100 (7/11)

- C++ compatibility (1/3)
  - For details, ask your system administrator.

No.	Target Machine	Item
1	[K computer/FX10/FX100]	Defaulting to the -Klib option
2	[FX100]	Change in the handling of options when the -Klto option is specified together with the -S or -Wa option
3	[K computer/FX10]	Changed default value for the level in the -Ksimd[=level] option
4	[K computer/FX10/FX100]	Changed specifications of the runtime information output function
5	[K computer/FX10/FX100]	LLVM OpenMP library installation
6	[K computer/FX10/FX100]	Deleted -gdwarf-2 option, and added -gdwarf and -gdwarf-4 options
7	[K computer/FX10/FX100]	Change of the standard template library (STL) in the C++03 specifications or C++11 specifications
8	[K computer/FX10/FX100]	Deleted compile options -stdlib and -Nstl
9	[K computer/FX10/FX100]	Deleted compile option -K{stl_fast_new|nostl_fast_new}
10	[K computer/FX10/FX100]	Changed default value for the compile option -f{signed-char|unsigned-char}
11	[K computer/FX10/FX100]	Changed values for predefined version macros
12	[K computer/FX10/FX100]	Deleted compile option -X (The language specification mode is GNU-compatible mode only.)
13	[K computer/FX10/FX100]	Changed GNU C++ compatible version
14	[K computer/FX10/FX100]	Changed default value for the compile option -std
15	[K computer/FX10/FX100]	Changed definitions of the macros __STRICT_ANSI__, linux, unix, and __cplusplus
16	[K computer/FX10/FX100]	Changed behavior when the compile options -Dname and -Uname are specified together
17	[K computer/FX10/FX100]	Changed behavior when the compile option -P is specified
18	[K computer/FX10/FX100]	Changed default value for the compile option -N{line|noline}
19	[K computer/FX10/FX100]	Object compatibility when using OpenMP
20	[K computer/FX10/FX100]	Deleted compile option -K{openmp_tls|openmp_notls}
21	[K computer/FX10/FX100]	Deleted options specific to the SPARC instruction set architecture
22	[K computer/FX10/FX100]	Deleted GNU C++ compatible options specific to the SPARC instruction set architecture

A64FX Compatibility With K Computer/FX10/FX100 (8/11)

- C++ compatibility (2/3)
  - For details, ask your system administrator.

No.	Target Machine	Item
23	[K computer/FX10/FX100]	Deleted compile options specific to HPC-ACE and HPC-ACE2
24	[FX100]	Changed specifications dependent on the instruction set architecture
25	[K computer/FX10/FX100]	Changed specifications of options related to -Kstrict_aliasing
26	[K computer/FX10/FX100]	Deleted -Nuse_rodata option, and changed inductive relationship
27	[K computer/FX10/FX100]	Changed options induced from the compiler options -O2 and -O3
28	[K computer/FX10/FX100]	Changed -K{XFILL|NOXFILL} option, optimization specifier {XFILL|NOXFILL}, and optimization information output
29	[K computer/FX10/FX100]	Deleted -K{shortloop=N|noshortloop} option and optimization specifier {SHORTLOOP(n)|NOSHORTLOOP}
30	[K computer/FX10/FX100]	Deleted -K{uxsimd|nouxsimd} option and optimization specifier {UXSIMD|NOUXSIMD}
31	[K computer/FX10/FX100]	Changed specifications of the -Kswp and -Kswp_strong options
32	[K computer/FX10/FX100]	Changed display format of compiled information (SIMDization)
33	[K computer/FX10/FX100]	Changed display format of compiler messages (loop fission)
34	[K computer/FX10/FX100]	Changed display format of compiled information (loop fission)
35	[K computer/FX10/FX100]	Changed display format of compiled information (software pipelining)
36	[K computer/FX10/FX100]	Deleted -Kdalign option
37	[K computer/FX10/FX100]	Changed specifications of the built-in functions __sync_fetch_and_nand and __sync_nand_and_fetch
38	[K computer]	Changed deduction relationship between the optimization level and the compile option -Kloop_fission
39	[K computer/FX10/FX100]	Changed compile option -Kloop_part_simd and optimization specifier LOOP_PART_SIMD
40	[K computer/FX10/FX100]	Changed compile option -Kloop_part_parallel and optimization specifier LOOP_PART_PARALLEL
41	[K computer/FX10/FX100]	Deleted compile option -K{loop_fission_if|loop_nofission_if}
42	[K computer/FX10/FX100]	Change of -Kprefetch_strong to default
43	[K computer/FX10/FX100]	Requirement that -Khpctag be enabled for -Kprefetch_nostrong and -Kprefetch_nostrong_L2
44	[K computer]	Deleted compile options specific to the K computer

A64FX Compatibility With K Computer/FX10/FX100 (9/11)

- C++ compatibility (3/3)
  - For details, ask your system administrator.

No.	Target Machine	Item
45	[FX10]	Deleted compile options specific to the FX10
46	[FX100]	Defaulting to the SIMDization function through multiple redundant executions with the SIMD length, and deleted optimization specifier simd_redundant_vl
47	[K computer/FX10/FX100]	Modified runtime messages
48	[K computer/FX10/FX100]	Changed behavior when a specified option is unrecognizable
49	[K computer/FX10/FX100]	Changed options induced from the compile option -Kfast
50	[K computer/FX10/FX100]	Runtime messages that are output during parallel processing
51	[K computer/FX10/FX100]	Runtime messages that are output when capturing floating-point exceptions
52	[K computer/FX10/FX100]	Changed Japanese message for jwd8205o-i

A64FX Compatibility With K Computer/FX10/FX100 (10/11)

- MPI compatibility (1/2)
  - For details, ask your system administrator.

No.	Target Machine	Item
1	[K computer/FX10/FX100]	Deleted Hasty Rendezvous communication function
2	[K computer/FX10/FX100]	Deleted MCA parameter dpm_ple_socket_timeout
3	[K computer/FX10/FX100]	Deleted extension RDMA interface
4	[K computer/FX10/FX100]	Changed rank query interface
5	[K computer/FX10/FX100]	Changed output of error messages related to the mpiexec command
6	[K computer/FX10/FX100]	Changed specifications for output of an error message when the mpiexec command is executed with no option specified
7	[K computer/FX10/FX100]	Changed error message for insufficient memory
8	[K computer/FX10/FX100]	Changed and deleted error messages
9	[K computer/FX10/FX100]	Changed warning messages
10	[K computer/FX10/FX100]	Deleted signal SIGCHLD
11	[K computer/FX10/FX100]	MPI functions in C no longer hooked, since the profiling interface of Fortran is used
12	[K computer/FX10/FX100]	Change of the MCA parameter orte_abort_print_stack to opal_abort_print_stack
13	[K computer/FX10/FX100]	Changed restrictions when generating processes from within MPI programs
14	[FX100]	Changed default value for the MCA parameter common_tofu_packet_mtu from 1792 to 1920
15	[K computer/FX10/FX100]	Changed error message [mpi::common-tofu::tofu-signal-*]
16	[FX100/FX10]	Deleted -nompi option of the mpiexec command
17	[FX100/FX10]	Changed name of the MCA parameter dpm_ple_no_establish_connection
18	[K computer/FX10/FX100]	Changed algorithm of the reduction operation used for group communication
19	[K computer/FX10/FX100]	Changed default value for the MCA parameter coll_tbi_use_on_max_min
20	[K computer/FX10/FX100]	Changed conditions for applying the barrier communication function in the MPI_REDUCE routine and MPI_ALLREDUCE routine
21	[K computer/FX10/FX100]	Deleted MCA parameters coll_tuned_use_6d_algorithm and coll_tuned_scatterv_use_linear_sync
22	[FX100]	Changed behavior when the specified value for the MCA parameter opal_progress_thread_mode is invalid (other than 0 to 3)

A64FX Compatibility With K Computer/FX10/FX100 (11/11)

- MPI compatibility (2/2)
  - For details, ask your system administrator.

No.	Target Machine	Item
23	[K computer]	Changed MCA parameter names and error messages related to the communication timeout setting function
24	[FX10/FX100]	Deleted MCA parameters mpi_deadlock_timeout and mpi_deadlock_timeout_delay

- Math library compatibility
  - For details, ask your system administrator.

No.	Target Machine	Item
1	[FX100]	Update of the BLAS header file cblas.h to the latest version

MPI Extension Functions and Data Types

- MPI Extension Functions and Data Types

MPI Extension Functions and Data Types

- Extension functions

Function Name	A64FX Behavior	FX700 Behavior
FJMPI_Collection_*	Interval-specified MPI statistical information	Neither collects information nor displays results.
FJMPI_Progress_*	Promotes communication using assistant cores.	Does not allow communication and computing to overlap. (Same as not being called)
FJMPI_Prequest_*	Extended persistent communication request	Works as a wrapper for MPI_Send_init, etc.
FJMPI_Topology_*	Obtains Tofu coordinates.	Returns a value that simulates a 1-dimensional Tofu job, based on the rank number.
FJMPI_Mswk_*	Master-worker function	Same as the A64FX
MPIX_*_init	Preliminarily implements persistent collective communication requests.	Same as the A64FX

- Data types
  - Half-precision data types are added.

Endian

- Endian

Endian

- Endian

Node Type	K Computer	A64FX
Compute node	Big endian	Little endian
Login node (IA server)	Little endian	Little endian

- Automatic conversion during execution (Fujitsu Fortran)
  - Using runtime options (-Wl,-Tu_num) in Fujitsu Fortran, you can input and output big-endian data.
  - Only one device number can be specified. If the device number is omitted, the target is all device numbers connected to the unformatted file.

Example: Runtime option (Device number = 10)

```
./a.out  -Wl,-T10
```

Example: Environment variable (Device number = 10)

```
export FORT90L="-Wl,-T10"
./a.out
```

Acceleration With Compiler

- Prefetch
- SIMDization
- Software Pipelining
- Loop Optimization and Instruction Scheduling
- Optimization for Loops
- Automatic Parallelization

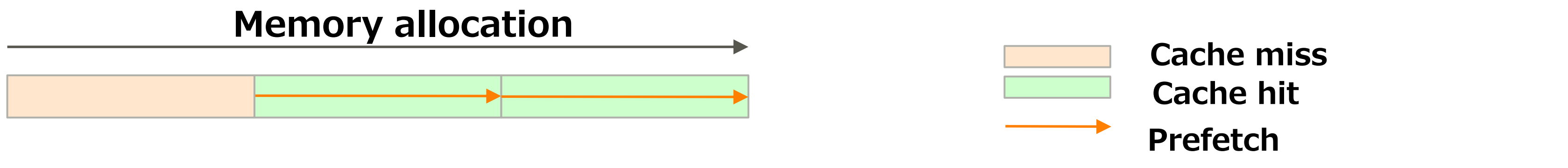
Prefetch

- About Prefetch
- Hardware Prefetch
- Software Prefetch
- Harmonization of Prefetch

About Prefetch

What is Prefetch?

Prefetch is a mechanism that raises performance by loading data into the cache before the data is required by an executed instruction.

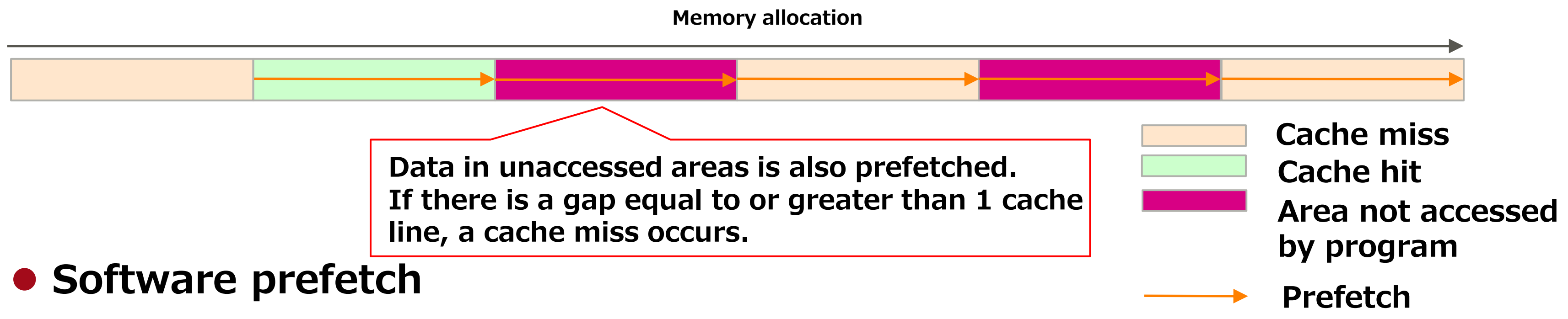


The two types of prefetch are hardware prefetch and software prefetch.

- **Hardware prefetch**

Hardware prefetches data by predicting data access based on the regularity of memory access by programs.

The cache efficiency of a program may degrade significantly because data is also prefetched from areas not accessed by the program. In such cases, use software prefetch.


Memory allocation
Data in unaccessed areas is also prefetched.
If there is a gap equal to or greater than 1 cache line, a cache miss occurs.
Cache miss
Cache hit
Area not accessed by program
Prefetch

- **Software prefetch**

Software (compiler) prefetches data by analyzing programs and generating a prefetch instruction.

About Prefetch (1/2)

- The A64FX supports the following new functions as hardware and software prefetch:
  - Hardware prefetch (HWPF)
    - Hardware prefetch distance setting function
    - Hardware stride prefetch function
  - Software prefetch (SWPF)
    - Automatic adjustment of the prefetch distance
    - SVE Gather prefetch instruction support

- By using these prefetch functions, you can mask data access latency to speed up application execution. (Latency masking)

About Prefetch (2/2)

- Prefetch distance
  Hardware prefetch and software prefetch perform data prefetching on the lines ahead as indicated below.

	Hardware Prefetch		Software Prefetch	
	L1 Prefetch	L2 Prefetch	L1 Prefetch	L2 Prefetch
FX100	2 lines	Up to 16 lines	3 lines	15 lines
A64FX	**Up to 6 lines**	**Up to 40 lines**	**Automatic**	**Automatic**

The hardware prefetch distance can also be set by users.

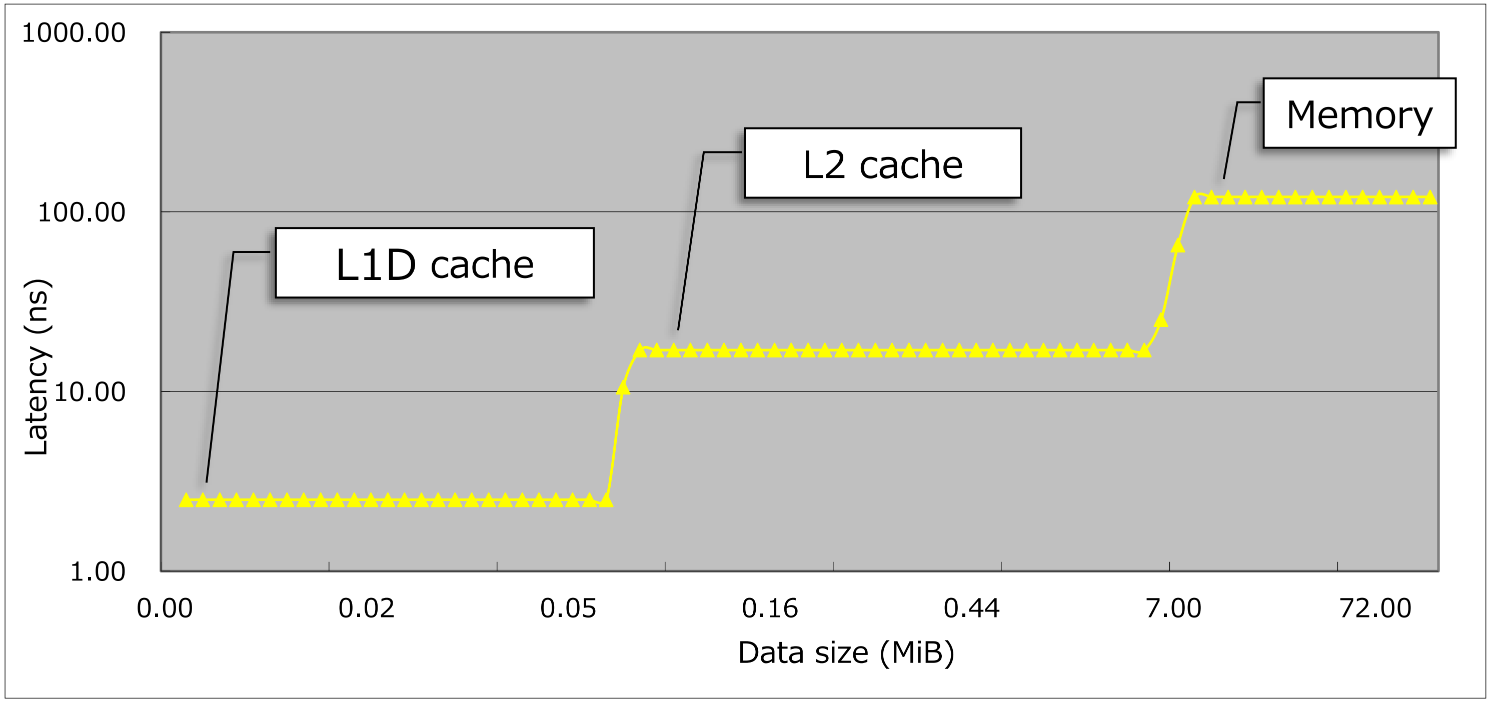
The distance is automatically adjusted for software prefetch.

- When tuning loop blocking or outer loop unrolling, be aware that <u>the hardware prefetch may not work</u> if the inner loop length decreases.

[Reference] What is Latency Masking?

- Latency masking hides data access latency (period from a data transmission request until a response returns) by using prefetch. There are three types of data access: L1D cache, L2 cache, and memory. The prefetch target for latency masking is the L2 cache and memory.

- Data access latency measurement results from LMbench (at integer access)



Hardware Prefetch

- Hardware Prefetch Behavior
- Hardware Prefetch Distance Setting Function
- Hardware Stride Prefetch Function

Hardware Prefetch Behavior

- Conditions of hardware prefetch behavior
  - Cause: L1 miss
  - L1 miss that proceeds sequentially by cache line

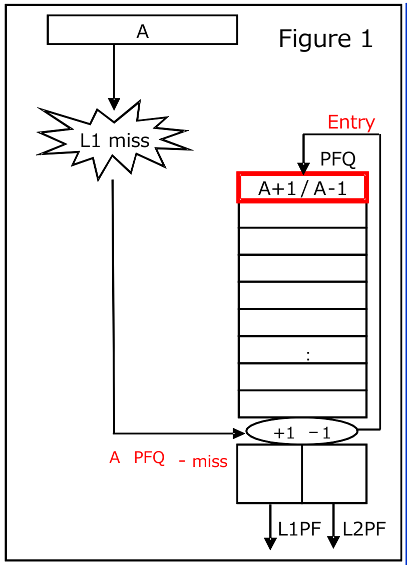
- Algorithm of hardware prefetch behavior

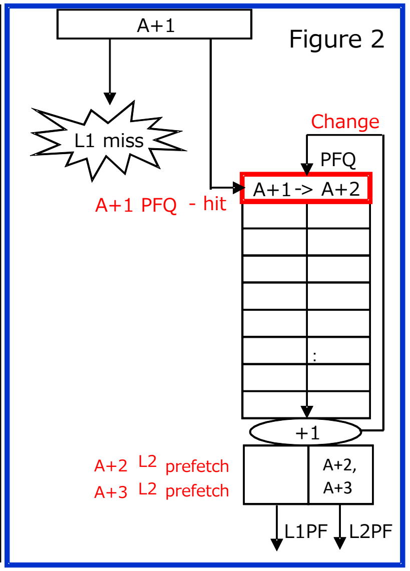
1. A miss occurs on cache line A, resulting in registration of the lines A+1/A-1 as a new entry in a 16-entry FIFO queue called PFQ (prefetch queue). (Figure 1)
2. When the line A+1 is accessed in subsequent access, the result is a hit of the registered A+1 entry in PFQ, and the access is determined as ascending stream access. Then, HWPF begins in ascending order. Also, A+1 in PFQ is updated to A+2. (Figure 2)
3. After the above-described detection of stream access, L2 prefetch fetches 2 lines of data at a time to extend to 40 lines ahead. In the A64FX, L1 prefetch also fetches 2 lines of data at a time, like L2 prefetch, to extend to 6 lines ahead. (Figure 3)

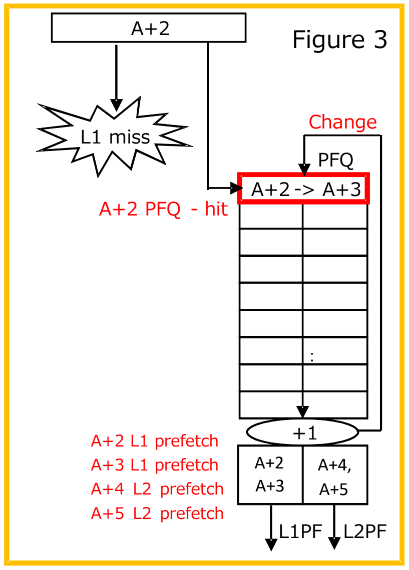
Functional extension on A64FX
(fixed distance per line on FX100)

Note: The numbers in the table represent lines. In byte representation, read 1 line as equal to 256 bytes.

- Outlines of prefetch







- Correspondence between PFQ and prefetch addresses

PFQ-hit	L2HWPF-ADRS	L1PHWF-ADRS
A+ 1	A+ 2, A+ 3	
A+ 2	A+ 4, A+ 5	A+ 2, A+ 3
A+ 3	A+ 6, A+ 7	A+ 4, A+ 5
A+ 4	A+ 8, A+ 9	A+ 6, A+ 7
A+ 5	A+10, A+11	A+ 8, A+ 9
A+ 6	A+12, A+13	A+10
A+ 7	A+14, A+15	A+11
A+ 8	A+16, A+17	A+12
A+ 9	A+18, A+19	A+13
A+10	A+20	A+14
A+11	A+21	A+15
A+12	A+22	A+16
A+13	A+23	A+17
A+14	A+24	A+18
A+15	A+25	A+19

* L2PF=10 lines ahead, L1PF=4 lines ahead

Hardware Prefetch Distance Setting Function

- Command (hwpfctl) for setting the hardware prefetch distance

Item	Details

Format	hwpfctl [--disableL1] [--disableL2] [--distL1 lines_l1] [--distL2 lines_l2] [--weakL1] [--weakL2] [--verbose] command {arguments ...}
	hwpfctl --default [--verbose] command {arguments ...}
	hwpfctl --reset [--verbose]
	hwpfctl –-help

Description	The hwpfctl command changes the behavior of the hardware prefetch (stream detect mode) mechanism mounted on the A64FX. The CPU core to be changed is chosen based on process affinity.
Option	--disableL1 --disableL2 Disables hardware prefetch on the L1/L2 cache. If omitted, hardware prefetch is enabled. --distL1=lines_l1 --distL2=lines_l2 Specifies the cache lines to prefetch on the L1/L2 cache. The prefetch of this number of cache lines starts on the cache line where a cache miss occurred. In lines_l1, you can specify a value from 1 to 15 lines to be prefetched on the L1 cache. In lines_l2, you can specify a value from 1 to 60 lines to be prefetched on the L2 cache. However, the value specified in lines_l2 is rounded up to a multiple of 4, which is then written to the system register. If 0 is specified, the behavior follows the default value of the CPU. If omitted or the specified value is invalid, 0 is assumed specified. --weakL1 --weakL2 Specifies "weak" for the priority of prefetch requests to the L1/L2 cache. If omitted, "strong" is assumed. --default Runs the command with the default settings. Options other than --verbose are ignored. --reset Initializes the system register values. Options other than --verbose are ignored. --verbose Outputs the system register values before and after changes. --help Displays instructions.

- Example of using the command (hwpfctl) for setting the hardware prefetch distance

```
hwpfctl  –distL1=6  –distL2=40  a.out
```

Hardware Stride Prefetch Function

- The A64FX supports the hardware stride prefetch function.
  - This function is an extension of the -Kprefetch_stride option.

-K{prefetch_stride[={soft|hard_auto|hard_always}]|prefetch_nostride}

The option specifies whether to prefetch the array data accessed with an even longer stride than the line size of the cache used in a loop.

This includes loops with the addresses for prefetching not yet determined at the compile time.

If the -Kprefetch_stride option is specified, the function is executed. If omitted, the -Kprefetch_nostride option is applied.

If ={soft|hard_auto|hard_always} is omitted from -Kprefetch_stride, the -Kprefetch_stride=soft option is applied.

The -Kprefetch_stride=soft option and -Kprefetch_nostride option are valid when the -O1 option or higher is enabled.

The -Kprefetch_stride=hard_auto option and -Kprefetch_stride=hard_always option are valid when the -Khpctag option and -O1 option or higher are enabled.

-Kprefetch_stride=soft
  Generate a prefetch instruction for prefetching.

-Kprefetch_stride=hard_auto
  Use the hardware stride prefetcher for prefetching.
  If this option is specified, the hardware stride prefetcher is set to prefetch only data that is not on the cache.

-Kprefetch_stride=hard_always
  Use the hardware stride prefetcher for prefetching.
  If this option is specified, the hardware stride prefetcher is set to always prefetch data, unlike the -Kprefetch_stride=hard_auto option.

Software Prefetch

- Automatic Adjustment of Prefetch Distance
- [Reference] Prefetch Instructions Provided by A64FX
- Prefetch Options
- Prefetch Optimization Specifiers (OCL)

Automatic Adjustment of Prefetch Distance

FUJITSU

- What is automatic adjustment of the prefetch distance?
  - In general, the prefetch distance may be as long as the possible extent of latency masking.
  - However, a longer distance than required has the following negative effects:
    - Iterations near the start of loops that do not prefetch the referenced areas
    - Increase in the number of lines that should be cached concurrently
  - The appropriate prefetch distance varies by loop, according to the execution period and cache usage per iteration of the loop. Until now, if not specified by the user, the prefetch distance uses a longer fixed value, which could cause the above-described negative effects.

The A64FX heuristically determines the appropriate prefetch distance for every loop, taking the following conditions into consideration, when no value is specified by the user:

- **Number of streams**
- **Loop length**

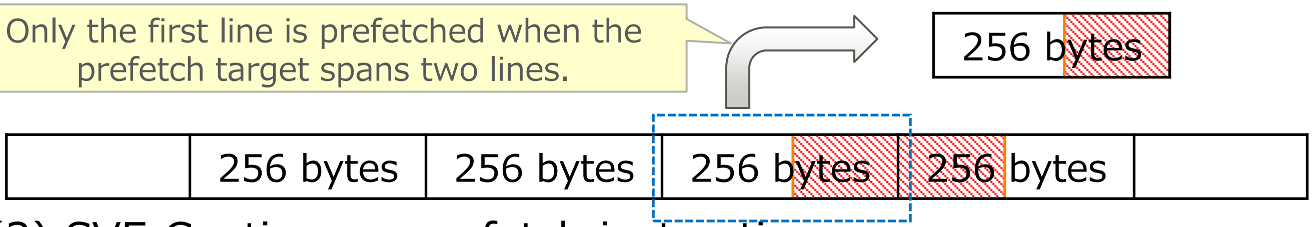
* The function for automatic adjustment of the prefetch distance works when the prefetch distance is not explicitly specified.

[Reference] Prefetch Instructions Provided by A64FX

- Prefetch instructions provided by the A64FX
  - (1) ARMv8 prefetch instruction
    - Prefetch instruction supporting sequential load/store instructions (no consideration of prefetch across lines)

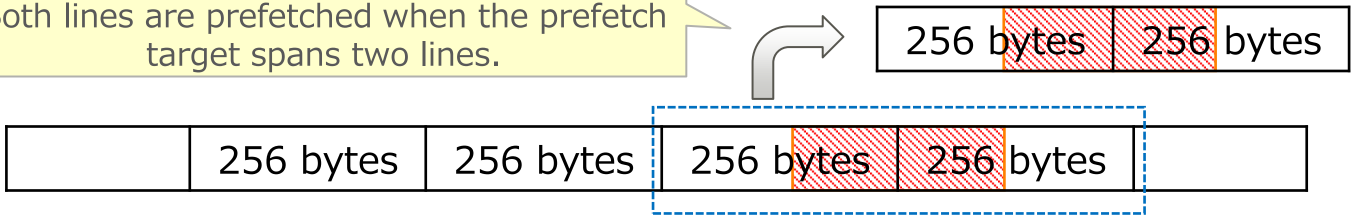
Instruction used for sequential access:
(1) ARMv8 prefetch instruction, and
(2) SVE Contiguous prefetch instruction

Only the first line is prefetched when the prefetch target spans two lines.



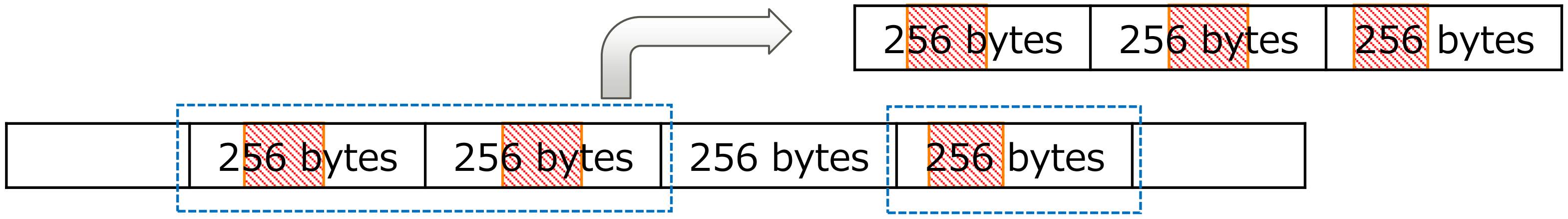
  - (2) SVE Contiguous prefetch instruction
    - Prefetch instruction supporting sequential load/store instructions (consideration of prefetch across lines)

Both lines are prefetched when the prefetch target spans two lines.



  - (3) SVE Gather prefetch instruction
    - Prefetch instruction supporting scattered access instructions (Gath

Instruction used for indirect access:
(3) SVE Gather prefetch instruction



Prefetch Options (1/3)

Option Format	Functional Outline
-Knoprefetch	Specifies not to generate objects using a prefetch instruction.
-Kprefetch_cache_level = { 1 | 2 | all }	Specifies which cache level to prefetch data to.
-K{ prefetch_sequential[=kind] | prefetch_nosequential } kind: { auto | soft }	Specifies whether to generate objects using a prefetch instruction for the array data used in a loop and accessed sequentially.
-K{prefetch_stride [={soft|hard_auto|hard_always}] |prefetch_nostride}	Specifies whether to prefetch the array data accessed with an even longer stride than the line size of the cache used in a loop. This includes loops with the addresses for prefetching not yet determined at the compile time. If the -Kprefetch_stride option is specified, the function is executed.
-K{ prefetch_indirect | prefetch_noindirect }	Specifies whether to generate objects using a prefetch instruction for the array data used in a loop and accessed indirectly (list access). (This option is valid when the -O1 or higher option is enabled.)
-K{ prefetch_conditional | prefetch_noconditional }	Specifies whether to generate a prefetch instruction for the array data used in the blocks included in an IF construct or CASE construct.
-K{ prefetch_infer | prefetch_noinfer }	Specifies whether to generate a prefetch instruction for sequential access even if the prefetch distance is not known.

Prefetch Options (2/3)

Option Format	Functional Outline
-K{ prefetch_strong | prefetch_nostrong }	Specifies whether to generate a strong prefetch instruction when generating a prefetch instruction for the primary cache.
-K{ prefetch_strong_L2 | prefetch_nostrong_L2 }	Specifies whether to generate a strong prefetch instruction when generating a prefetch instruction for the secondary cache.
-Kprefetch_iteration=N 1 ≦N ≦ 10000	Specifies the target as the data referenced or defined after N cycles of a loop, when generating a prefetch instruction. For a loop with SIMDization applied, specify N as the number of loop iterations after SIMDization.
-Kprefetch_iteration_L2=N 1 ≦N ≦ 10000	Specifies the target as the data referenced or defined after N cycles of a loop, when generating a prefetch instruction. For a loop with SIMDization applied, specify N as the number of loop iterations after SIMDization.
-Kprefetch_line=N 1 ≦N ≦ 100	Specifies the target as the corresponding data prefetched N cache lines ahead, when generating a prefetch instruction .
-Kprefetch_line_L2=N 1 ≦N ≦ 100	Specifies the target as the corresponding data prefetched N cache lines ahead, when generating a prefetch instruction.

Prefetch Options (3/3)

Option	Usage Example

-Kprefetch_indirect	Specify -Kprefetch_indirect in the following situation. Suppose you want to output prefetch instructions like the optimization control line shown below. Also, for list access, the array is written to the index like a(m(i)) in the program below. However, the option may cause execution performance to decrease depending on the cache efficiency of loops, the presence of branches, or the complexity of the index.
	`do i=1,N`
	`    /* Outputs L2 prefetch instruction on a(m(i+α)) */`
	`    /* Outputs L1 prefetch instruction on a(m(i+β)) */`
	`    b(i) = b(i) + a(m(i))`
	`enddo`

-Kprefetch_stride	Specify -Kprefetch_stride in the following situation. Suppose you want to output prefetch instructions like the following optimization control line. Also, in stride access, the increment value for loops may be so large that the array elements accessed in the current cycle and next cycle are not on the same cache line as shown in the program below. However, the option may cause execution performance to decrease depending on the cache efficiency of loops or the presence of branches.
	`do i=1,N,100`
	`    /* Outputs L2 prefetch instruction on a(i+100*a) */`
	`    /* Outputs L1 prefetch instruction on a(i+100*β) */`
	`    /* Outputs L2 prefetch instruction on b(i+100*a) */`
	`    /* Outputs L1 prefetch instruction on b(i+100*β) */`
	`    b(i) = a(i) + 1.0`
	`enddo`

Prefetch Optimization Specifiers (OCL) (1/2)

Optimization Specifier	Meaning	Specifiable in Following Units on Optimization Control Line?			
		Program	DO Loop	Stmt.	Array Assignment Stmt.
PREFETCH	Enables the automatic prefetch function of the compiler.	✓	✓	x	x
NOPREFETCH	Disables the automatic prefetch function of the compiler.	✓	✓	x	x
PREFETCH_CACHE_LEVEL (c-level)	Specifies c-level, the cache level of the data to be prefetched. c-level is 1, 2, or all.	✓	✓	x	✓
PREFETCH_INFER	Specifies that prefetch be output for sequential access even if the prefetch distance is not known.	✓	✓	x	✓
PREFETCH_NOINFER	Specifies that prefetch not be output for sequential access if the prefetch distance is not known.	✓	✓	x	✓
PREFETCH_ITERATION(n)	Specifies the target as the data referenced or defined after n cycles of a loop, when generating a prefetch instruction. n is a decimal number from 1 to 10,000.	✓	✓	x	x
PREFETCH_ITERATION_L2(n)	Specifies the target as the data referenced or defined after n cycles of a loop, when generating a prefetch instruction. However, this function targets only the prefetch instructions for secondary cache prefetching. n is a decimal number from 1 to 10,000.	✓	✓	x	✓

Prefetch Optimization Specifiers (OCL) (2/2)

FUJITSU

Optimization Specifier	Meaning	Specifiable in Following Units on Optimization Control Line?			
		Program	DO Loop	Stmt.	Array Assignment Stmt.
PREFETCH_READ (name[,level={1|2}] [,strong={0|1}])	Specifies that prefetch instructions be generated for the referenced data. "name" is the name of an array element, "level" is the cache level for prefetching, and "strong" indicates whether to use strong prefetch.	x	x	✓	x
PREFETCH_WRITE (name[,level={1|2}] [,strong={0|1}])	Specifies that prefetch instructions be generated for the defined data. "name" is the name of an array element, "level" is the cache level for prefetching, and "strong" indicates whether to use strong prefetch.	x	x	✓	x
PREFETCH_SEQUENTIAL [(AUTO|SOFT)]	Specifies that prefetch instructions be generated for the array data accessed sequentially.	✓	✓	x	✓
PREFETCH_STRONG	Specifies that prefetch instructions for the primary cache be generated as strong prefetch.	✓	✓	x	✓
PREFETCH_NOSTRONG	Specifies that prefetch instruction for the primary cache not be generated as strong prefetch.	✓	✓	x	✓
PREFETCH_STRONG_L2	Specifies that prefetch instructions for the secondary cache be generated as strong prefetch.	✓	✓	x	✓
PREFETCH_NOSTRONG_L2	Specifies that prefetch instructions for the secondary cache not be generated as strong prefetch.	✓	✓	x	✓

Harmonization of Prefetch

- Harmonization of Hardware and Software Prefetch

Harmonization of Hardware and Software Prefetch

HWPF does not occur in the cases described below.
Complementation with SWPF can deliver optimum performance.

- Number of streams exceeds 16

  -> This is automatically identified by the compiler for processing done without the user's knowledge.
- Block stride cases
  - Loop blocking (Effective for array replacement, matrix operation, etc.)
  - Unrolling in the outer loop
- Access by masked SIMD

  When masked SIMD accesses a stream having an if statement, the true rate of the if statement may not allow sequential access. In this case, HWPF does not occur.

"Harmonization of Hardware and Software Prefetch" means complementation with SWPF in cases where HWPF does not occur.

Built-in Prefetch

- Built-in Prefetch

Built-in Prefetch

Built-in prefetch is supported in C/C++ clang mode.

- Specification method

```
__builtin_prefetch(*addr, rw, localy)
```

- *addr*: Specify the memory address to be prefetched.
- *rw*: Specify the type of prefetch.
  For read prefetch, specify 0. For write prefetch, specify 1.
- *localy*: Specify 0 to 3.
  For L1 cache prefetching, specify 3.
  For L2 cache prefetching, specify 2.

SIMD Vectorization (SIMDization) (<u>S</u>ingle <u>I</u>nstruction <u>M</u>ultiple <u>D</u>ata)

- <u>Basic Principles of SIMDization</u>
- <u>Example of SIMDization of Consecutive Data</u>
- <u>Confirmation of SIMDization</u>
- <u>SIMDizable Loops</u>
  - Assembler images from source code are used in descriptions of load/store and other instructions.

Basic Principles of SIMDization

- SIMD (single instruction multiple data)
  Parallel processing of multiple operations by a single instruction
- SIMD features of A64FX SVE
  Supports SIMD widths of up to 512 bits
  Supports variable-length SIMD widths
  Supports various accesses and operation instructions
  Can execute two multiply-add operation instructions simultaneously on one core

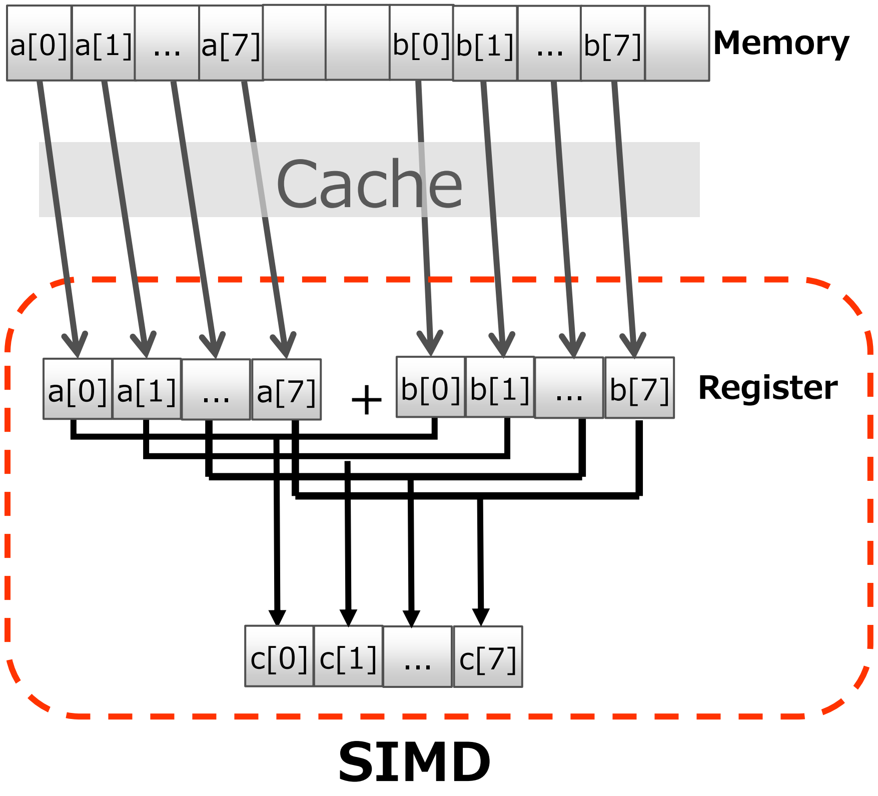
With double-precision:
A single core can process 16* operations simultaneously.
*8 SIMD width * 2 arithmetic pipeline

Can process 32 single-precision operations simultaneously
Can SIMDize integer types too

Application-friendly SIMD and faster computational processing realized

Program example:

```
for (i=0;i<7;++i) {
  c[ i ]=a[ i ]+b[ i ];
}
```



Example of SIMDization of Consecutive Data

- With data of the double-precision type

Source code

```
void foo(double a[N][N],
const double b[N][N], const double c[N][N])
{
    int i, j;
    for (i = 0; i < N; ++i) {
        for (j = 0; j < N; ++j) {
            a[ i ][ j ] = b[ i ][ j ] * c[ i ][ j ] ;
        }
    }
}
```

Not SIMDized

```
ldr     d1, [x15, x7, lsl #3]
ldr     x15, [x11]
fmul    d1, d1, d0
str     d1, [x15, x7, lsl #3]
```

→ Scalar instruction

SIMDized

```
ld1d    {z1.d}, p0/z, [x7, x1, lsl #3]
ld1d    {z0.d}, p0/z, [x4, x1, lsl #3]
fmul    z1.d, z0.d, z1.d
st1d    {z1.d}, p0, [x2, x1, lsl #3]
```

→ SIMD instruction

A single instruction processes data consisting of 8 elements, like b[i] [j] … b[i] [j+7].

For the innermost loop of the array, the 8 repeated array elements are SIMD ordered and the number of execution instructions is decreased to 1/8 to speed up processing.

Confirmation of SIMDization

- Example of compiled information output

```
(line-no.)(optimize)
     1              #define N 1000
     2              void foo(double a[N][N],const double b[N][N])
     3              {
     4                int i, j;
                    <<< Loop-information Sta
                    <<<  [PARALLELIZATION]
                    <<<    Standard iteration c
                    <<< Loop-information  En
     5  pp             for (i = 0; i < N; ++i) {
                    <<< Loop-information Start >>>
                    <<<  [OPTIMIZATION]
                    <<<    SIMD(VL: 8)
                    <<< Loop-information  End >>>
     6   p     2v        for (j = 0; j < N; ++j) {
     7   p     2v          a[ i ][ j ] = b[ i ][ j ] * 2.0;
     8   p     2v        }
     9                 }
    10              }
```

SIMDization information on loops
- v: SIMDized
- m: Includes both SIMDized and not-SIMDized parts
- s: Not SIMDized
- Blank: Not SIMDization target

SIMDized with 8-element vectors

SIMDization information on executable statements
- v: SIMDizable
- m: Includes both SIMDizable and non-SIMDizable parts
- s: Not SIMDizable

SIMDizable Loops

- Loop characteristics required in compiler SIMDization
  - The number of loop iterations can be determined before entering a loop. (*1)
  - No procedure/function is called inside a loop. (*2)
  - No more than one exit of the loop.
  - No I/O statements in the loop.
  - Data definition or order of data references order does not differ from serial execution.

*1 With SVE, this can also be a while loop.

*2 In some cases, SIMDization is possible through inline expansion.

Software Pipelinening

- Basic Principles of Software Pipelining
- Confirmation of Software Pipelining

Basic Principles of Software Pipelining

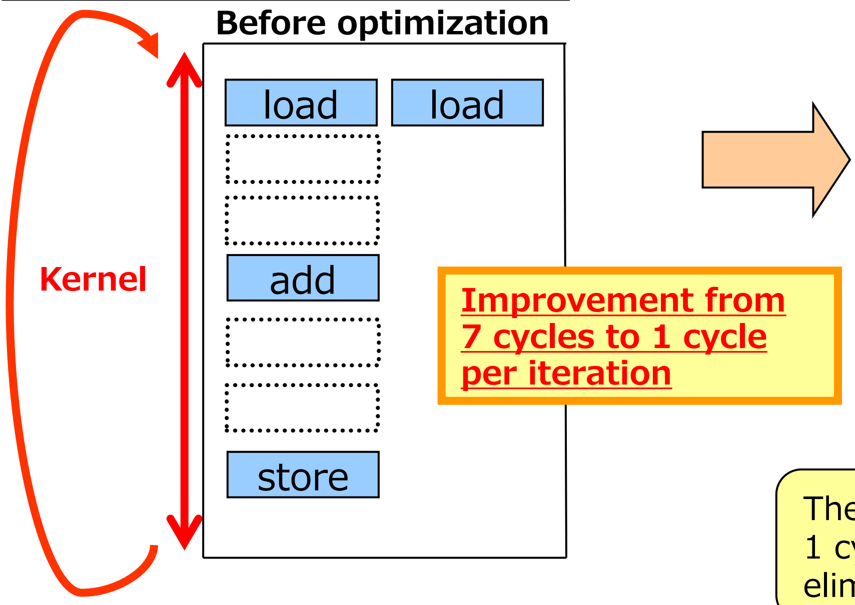
- Software pipelining improves instruction-level parallelism in a kernel loop by overlapping the current and next iterations of the loop.
  - The descriptions of concepts assume the following machine model.

The loop that best represents the features of the program is called a kernel loop.

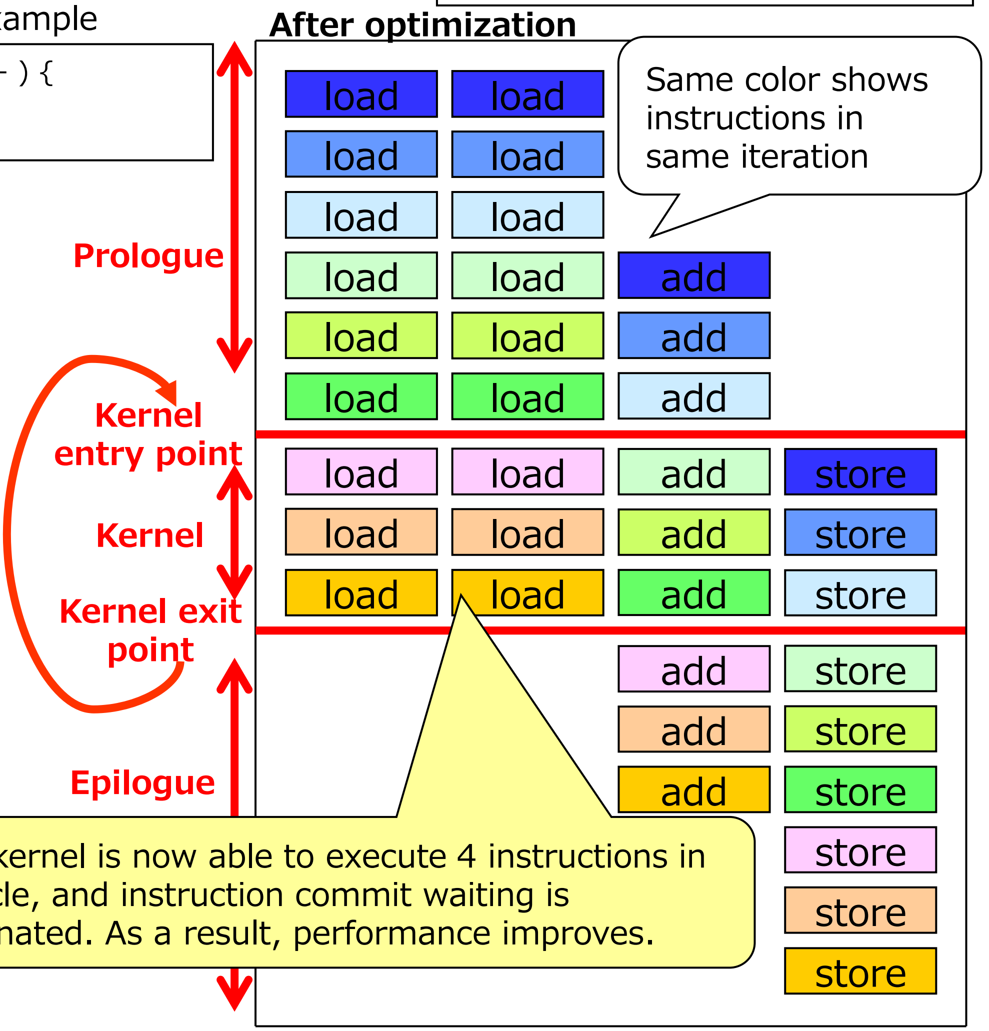
Latency of load = 3 cycles
Latency of add = 3 cycles
Latency of store = 1 cycle
Number of functional units for load/store = 3
Number of commits = 4 (Up to 3 can be issued concurrently for load/store)

Program example

```
for ( i=0 ; i < n ; i++ ) {
 a[i] = b[i] + c[i];
}
```



Improvement from 7 cycles to 1 cycle per iteration



The kernel is now able to execute 4 instructions in 1 cycle, and instruction commit waiting is eliminated. As a result, performance improves.

Confirmation of Software Pipelining

- Example of compiled information output

```
(line-no.)(optimize)
                         :
       <<< Loop-information Start >>>
       <<<  [PARALLELIZATION]
       <<<    Standard iteration count: 1067
       <<<  [OPTIMIZATION]
       <<<    SIMD(VL: 8)
       <<<    SOFTWARE PIPELINING(IPC: 3.00, ITR: 144, MVE: 4, POL: S)
              <<< Loop-information  End >>>
 6   s     2v    for (i = 0; i < N-1; ++i) {
 7   p     2v
 8   p     2v
```

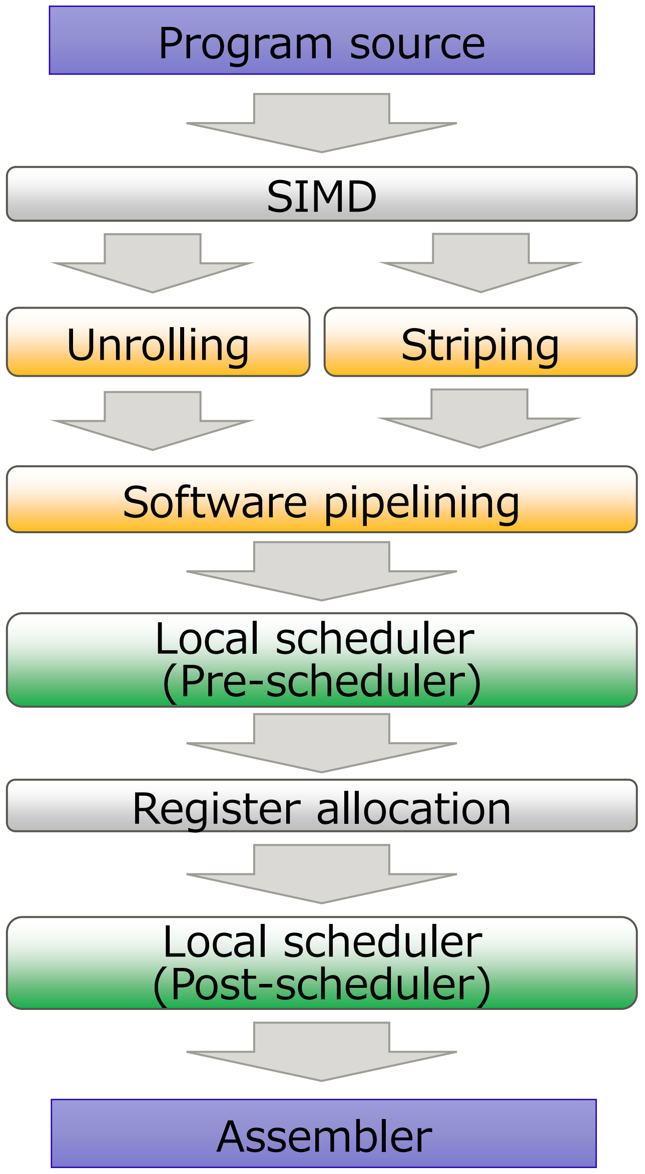
This indicates the loop was software pipelined.

SOFTWARE PIPELINING(IPC: ipc, ITR: itr, MVE: mve, POL: pol)

- Where software pipelining is applied to a loop, ipc is the predicted value of instructions per cycle for the loop.
- Where software pipelining is applied to a loop, which is selected at the execution time, itr is the required number of loop iterations for the loop. The same value is output in the compile message jwd8205o-i.
- mve is the number of loop expansions by software pipelining.
- pol is the applied instruction scheduling algorithm. `S` means that the algorithm fit for a small loop is applied. `L` means that the algorithm fit for a large loop is applied.

Loop Optimization and Instruction Scheduling

Order of Optimization



- Optimization order and descriptions
  - SIMD
    - SIMDize.
  - Unrolling
    - Optimize by expanding all of the executable statements in a loop by n times within the loop and reducing the number of loop cycles to 1/n.
  - Striping
    - Optimize by expanding a specific number (stripe length) of executable statements in a loop. Forcibly exchange executable statements.
  - Software pipelining
    - Optimize by improving instruction-level parallelism in a kernel loop by overlapping the current and next iterations of the loop.
  - Local scheduler (Pre-scheduler)
    - Exchange instructions (scheduling) with consideration of instruction latency and the live ranges of registers.
  - Register allocation
    - Allocate the registers used by variables and arrays.
  - Local scheduler (Post-scheduler)
    - Exchange instructions (scheduling) with consideration of the instructions added (spills) in register allocation.

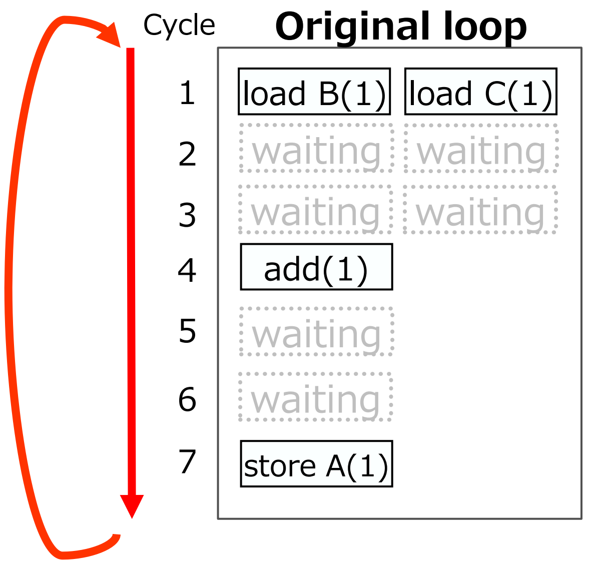
Unrolling Behavior

- Behavior of unrolling (without SWPL)
  - The descriptions of concepts assume the following machine model.

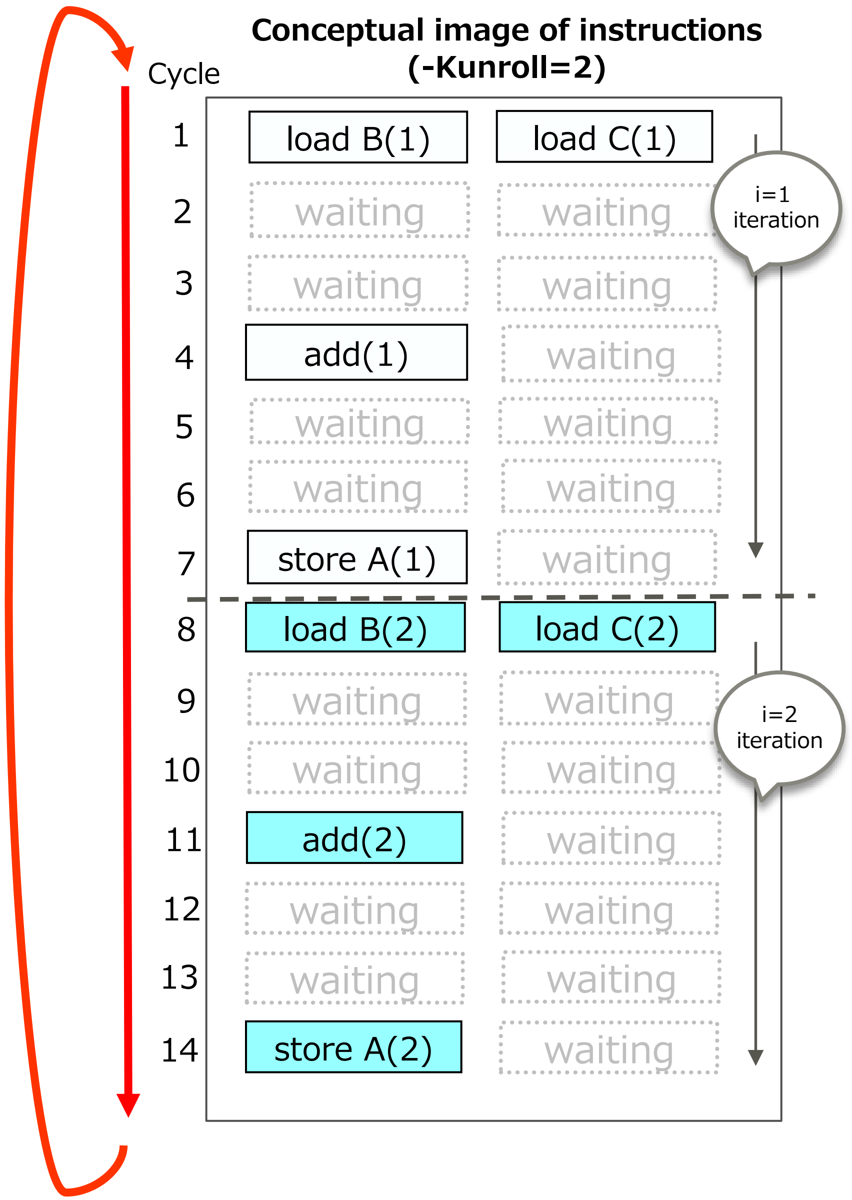
Latency of load	3 cycles
Latency of add	3 cycles
Latency of store	1 cycle
Number of load pipes	2
Number of store pipes	1
Number of concurrent instruction commits	4 instructions

Sample program

```
do i=1,n
   A(i) = B(i) + C(i)
enddo
```



Unrolling expands instructions in multiple iterations (without scheduling instructions)



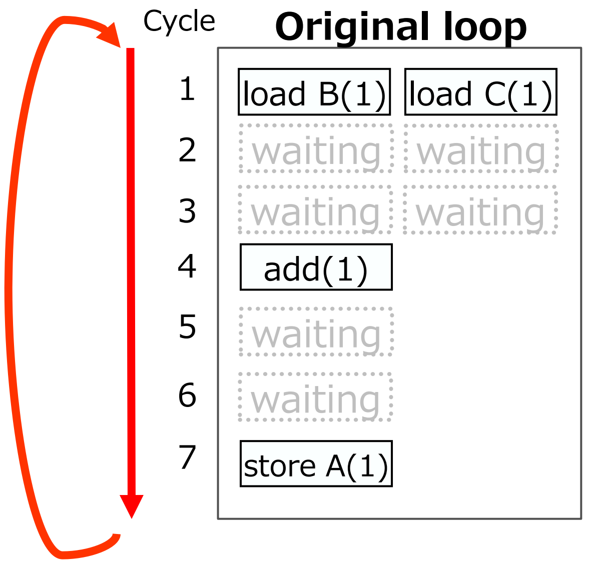
Striping Behavior

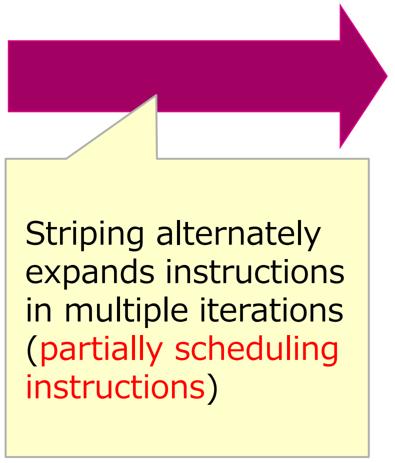
- Behavior of striping (without SWPL)
  - The descriptions of concepts assume the following machine model.

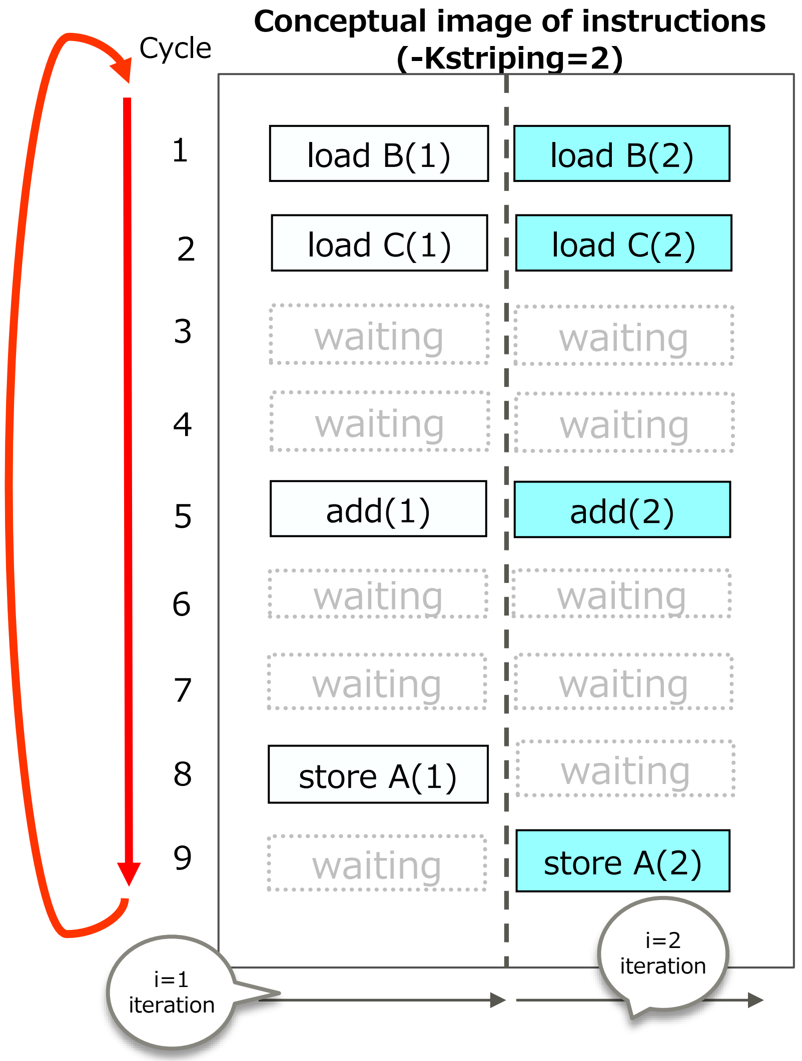
Latency of load	3 cycles
Latency of add	3 cycles
Latency of store	1 cycle
Number of load pipes	2
Number of store pipes	1
Number of concurrent instruction commits	4 instructions

Sample program

```
do i=1,n
  A(i) = B(i) + C(i)
enddo
```

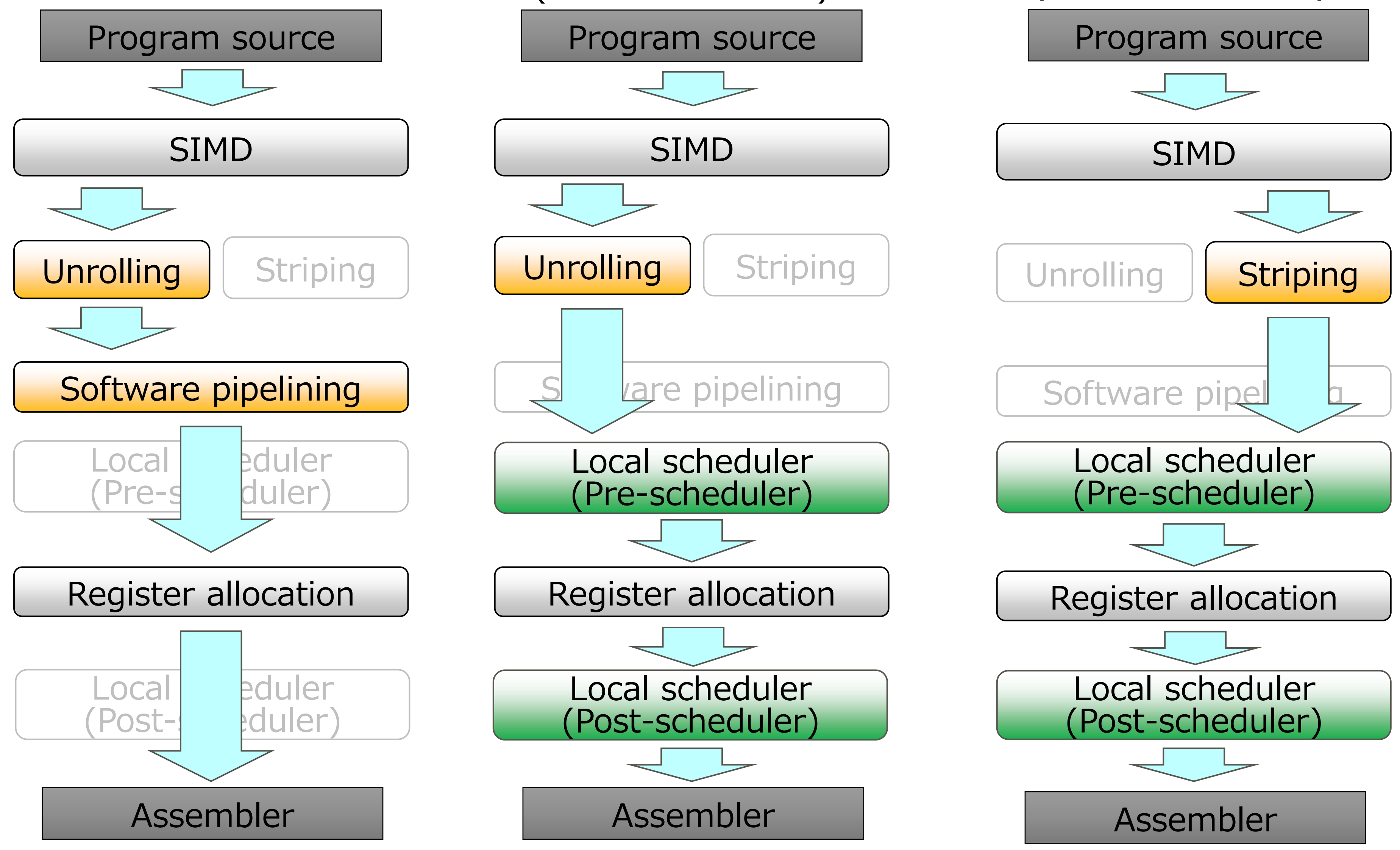



Striping alternately expands instructions in multiple iterations (partially scheduling instructions)



Flow of Loop Optimization and Control

- Software pipelining
- Unrolling (without SWPL) (+ local scheduler)
- Striping (without SWPL) (+ local scheduler)



Optimization for Loops

- About Loop Optimization
- Loop Interchange
- Loop Fusion
- Loop Unrolling
- Loop Collapse

About Loop Optimization

- Loop optimization
  - The table below lists typical loop optimization techniques automatically employed by the compiler.

Type of Loop Optimization/Transformation	Effect
Loop interchange	• Data localization • Parallelization of outer loops (Larger granularity)
Loop fusion	• Data localization • Improved instruction-level parallelism
Loop unrolling	• Fewer instructions • Improved instruction-level parallelism
Loop collapse	• Increased scheduling efficiency • Improved load balancing

Loop Interchange

- Purpose
  - Data localization
    - Cache utilization efficiency increases with the adoption of sequential access to array b.

Original Source

```
double a[N][M];
const double b[N][N][M];
void foo()
{
 int i, j, k;
 for (j = 0; j < M; ++j) {
   for (k = 0; k < N; ++k) {
     for (i = 0; i < N; ++i) {
       a[ k ][ j ] = a[ k ][ j ] + b[ k ][ i ][ j ];
     }
   }
 }
}
```

Stride access

Source Showing Compiler Optimization

```
double a[N][M];
const double b[N][N][M];
void foo()
{
 int i, j, k;
 for (k = 0; k < N; ++k) {
   for (i = 0; i < N; ++i) {
     for (j = 0; j < M; ++j) {
       a[ k ][ j ] = a[ k ][ j ] + b[ k ][ i ][ j ];
     }
   }
 }
}
```

Sequential access

Confirmation of Loop Interchange

- Example of compiled information output (Loop interchange)

```
(line-no.)(optimize)
                                        :
 <<< Loop-information Start >>>
              <<<  [OPTIMIZATION]
              <<<    INTERCHANGED(nest: 3)
              <<<    SIMD(VL: 8)
              <<<    SOFTWARE PIPELINING(IPC: 3.25, ITR: 192,
              <<< Loop-information  End >>>
 8   p      v    for (j = 0; j < M; ++j) {
              <<< Loop-information Start >>>
              <<<  [PARALLELIZATION]
              <<<    Standard iteration count: 2
              <<<  [OPTIMIZATION]
              <<<    INTERCHANGED(nest: 1)
              <<< Loop-information  End >>>
 9  pp             for (k = 0; k < N; ++k) {
              <<< Loop-information Start >>>
              <<<  [OPTIMIZATION]
              <<<    INTERCHANGED(nest: 2)
              <<< Loop-information  End >>>
10   p     2v        for (i = 0; i < N; ++i) {
11   p     2v          a[ k ][ j ] = a[ k ][ j ] + b[ k ][ i ][ j ];
12   p     2v        }
13                 }
14               }                    :
```

Loop interchange changed:
- loop j to nest 3,
- loop k to nest 1, and
- loop i to nest 2.

Loop Fusion

- Purpose
  - Data localization
    - Loop fusion enables reuse of array a.
  - Improved instruction-level parallelism
    - Loop fusion improves instruction-level parallelism by increasing the number of instructions in a loop and scheduling the instructions.

Original Source

```
void sub()
{
    int i;
    for (i = 0; i < N; ++i) {
        a[ i ] = b[ i ] + c[ i ];
    }
    for (i = 0; i < N; ++i) {
        d[ i ] = a[ i ] + e[ i ];
    }
}
```

Source Showing Compiler Optimization

```
void sub()
{
    int i;
    for (i = 0; i < N; ++i) {
        a[ i ] = b[ i ] + c[ i ];
        d[ i ] = a[ i ] + e[ i ];
    }
}
```

Confirmation of Loop Fusion

- Example of compiled information output (Loop fusion)

```
(line-no.)(optimize)
                    :
                      <<< Loop-information Start >>>
                 <<<  [PARALLELIZATION]
                 <<<    Standard iteration count: 728
                 <<<  [OPTIMIZATION]
                 <<<    FUSED(lines: 6,9)
                 <<<    SIMD(VL: 8)
                 <<<    SOFTWARE PIPELINING(IPC: 3.00, ITR: 144, MVE: 4, POL: S)
                 <<<    PREFETCH(HARD) Expected by compiler :
                 <<<      d, c, e, a, b
                 <<< Loop-information  End >>>
 6  pp     2v    for (i = 0; i < N; ++i) {
 7   p    2v      a[ i ] = b[ i ] + c[ i ];
 8   p    2v    }
                 <<< Loop-information Start >>>
                 <<<  [OPTIMIZATION]
                 <<<    FUSED
                 <<< Loop-information  End >>>
 9    p    2v    for (i = 0; i < N; ++i) {
10   p    2v      d[ i ] = a[ i ] + e[ i ];
11   p    2v    }
                    :
```

Loops on lines 6 and 9 fused

Loop Unrolling

- Purpose
  - Fewer instructions
    - Loop unrolling decreases the number of loop iterations, which in turn reduces branch instructions.
    - A general formula for reducing instructions is B[i+1]+b[i+2].
  - Improved instruction-level parallelism
    - Increasing the number of instructions per cycle increases the scheduling capacity and thus improves instruction-level parallelism.

Original Source	Source Showing Compiler Optimization

`#define N 10000`
`double a[N], b[N];`
`void sub()`
`{`
`    int i;`
`    for (i = 0; i < N-3; ++i) {`
`        a[ i ] = b[ i ] + b[ i+1 ] + b[ i+2 ];`
`    }`
`}`
	`#define N 10000`
	`double a[N], b[N];`
	`void sub()`
	`{`
	`    int i;`
	`    double t;`
	`    for (i = 0; i < N-3; i += 2) {`
	`        t = b[ i+1 ] + b[ i+2 ];`
	`        a[ i ] = b[ i ] + t;`
	`        a[ i+1 ] = t + b[ i+3 ];`
	`    }`
	`}`

Confirmation of Loop Unrolling

- Example of compiled information output (Loop unrolling)

```
(line-no.)(optimize)
                 :
       <<< Loop-information Start >>>
       <<<  [PARALLELIZATION]
       <<<    Standard iteration count: 843
       <<<  [OPTIMIZATION]
       <<<    SIMD(VL: 8)
       <<<    SOFTWARE PIPELINING(IPC: 2.57, ITR: 96, MVE: 2, POL: S)
             <<< Loop-information  End >>>
6  pp    2v    for (i = 0; i < N-3; ++i) {
7   p    2v      a[ i ] = b[ i ] + b[ i+1 ] + b[ i+2 ];
8   p    2v    }
                 :
```

Loop unrolled twice

Loop Collapse

- Purpose
  - <u>Increased software pipelining efficiency</u>
    - Loop collapse increases the number of loop iterations to increase software pipelining efficiency.
  - <u>Improved load imbalancing</u>
    - Loop collapse raises the likelihood of improved load balancing independently of the M value. For example, parallelization outside of the original source may have an adverse effect when the M value is 1, 2, or so on.

Original Source

```
double a[M][N], c;
void sub()
{
  int i,j;
  c=1.0;
  for (j = 0; j < M; ++j) {
    for (i = 0; i < N; ++i) {
      a[j][i] = a[j][i] + c;
    }
  }
}
```

Source Showing Compiler Optimization

```
double a[M*N], c;
void sub()
{
  int i,j;
  c=1.0;
  for (ij = 0; ij < N*M; ++ij) {
    a[ij] = a[ij] + c;
  }
}
```

Confirmation of Loop Collapse

- Example of compiled information output (Loop collapse)

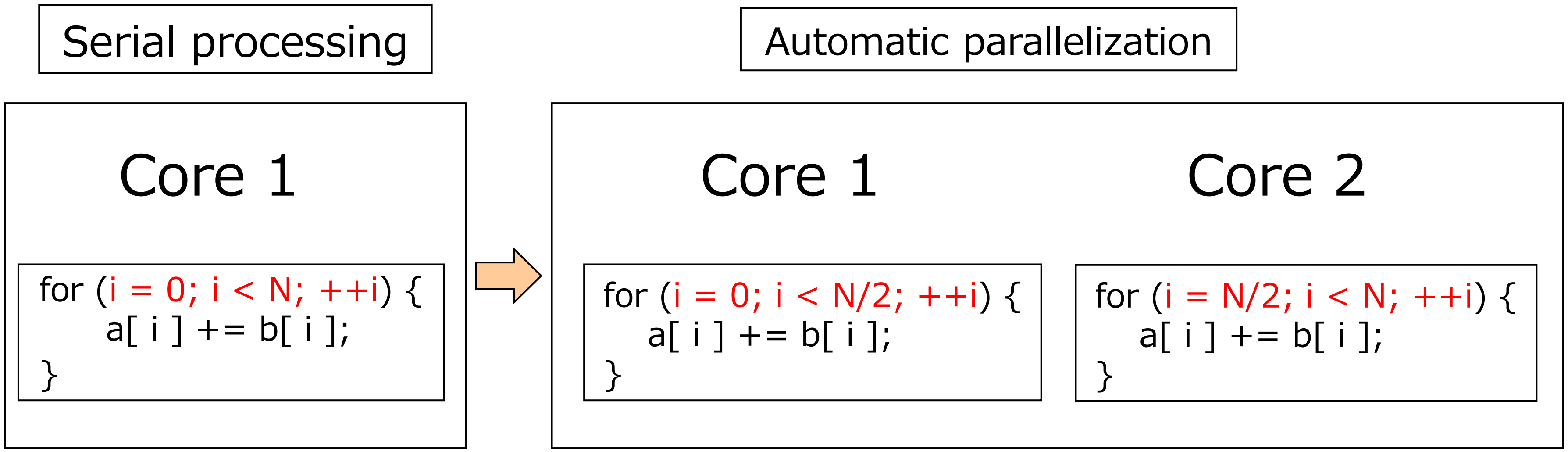
```
(line-no.)(optimize)
                        :
                       <<< Loop-information Start >>>
                  <<<  [PARALLELIZATION]
                  <<<    Standard iteration count: 1231
                  <<<  [OPTIMIZATION]
                  <<<    COLLAPSED
                  <<<    SIMD(VL: 8)
                  <<<    SOFTWARE PIPELINING(IPC: 2.25, ITR: 192, MVE: 7, POL: S)
                  <<< Loop-information  End >>>
 8  pp     2v    for (j = 0; j < M; ++j) {
                  <<< Loop-information Start >>>
                  <<<  [OPTIMIZATION]
                  <<<    COLLAPSED
                  <<< Loop-information  End >>>
  9   p     2v      for (i = 0; i < N; ++i) {
10   p     2v        a[j][i] = a[j][i] + c;
11   p     2v      }
12                }                          :
```

Loop collapsed

Automatic Parallelization

Simple Loop Slice

- Parallelization follows the allocation of loop indexes to multiple threads.



```
for (i = 0; i < N; ++i) {
     a[ i ] += b[ i ];
}
```

```
for (i = 0; i < N/2; ++i) {
     a[ i ] += b[ i ];
}
```

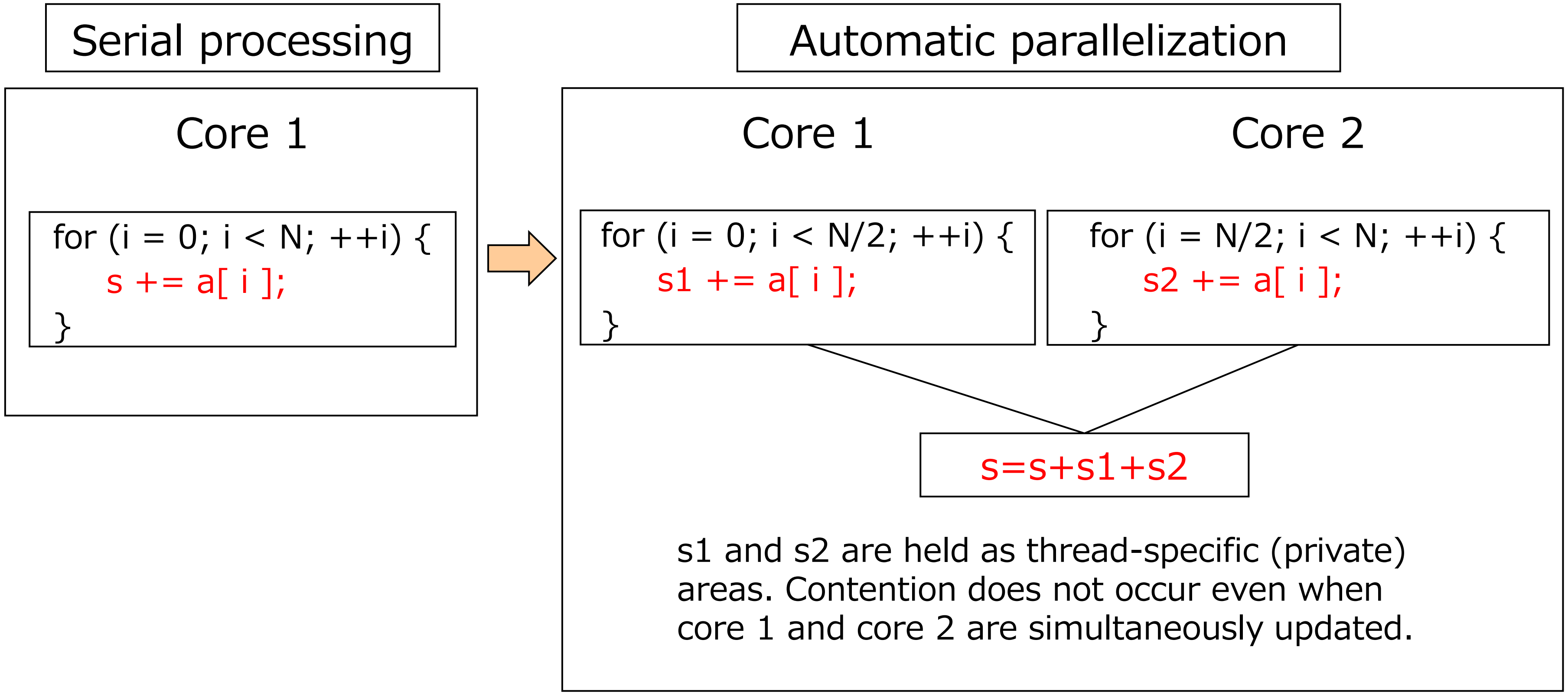
```
for (i = N/2; i < N; ++i) {
     a[ i ] += b[ i ];
}
```

(Example with 2 threads)

- An environment variable (PARALLEL) specifies the number of parallel threads.

Loop Slice by Reduction

- Parallelization for preventing data contention



```
for (i = 0; i < N; ++i) {
    s += a[ i ];
}
```

```
for (i = 0; i < N/2; ++i) {
    s1 += a[ i ];
}
```

```
for (i = N/2; i < N; ++i) {
    s2 += a[ i ];
}
```

s1 and s2 are held as thread-specific (private) areas. Contention does not occur even when core 1 and core 2 are simultaneously updated.

Note: A computation error may occur because the operation order differs from the sequential order.

If -Knoeval or –Knoreduction option is specified, parallelization is not possible.

Determination of Automatic Parallelization Possibility

- The parallelization targets exclude the following cases:
  - (1) Loop with a low cost identified at the compile time
  - (2) Loop with a low cost identified at the execution time
    - * The compiler outputs dynamic control for parallelization only when the loop cost is high.

(1)

```
double a[N][N], b[N][N];
void sub() {
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            a[ i ][ j ] = b[ i ][ j ] + 1.0;
        }
    }
}
```

(2)

```
extern double a[3][3], b[3][3];
void sub(int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            a[ i ][ j ] = b[ i ][ j ]+1.0;
        }
    }
}
```

Verification of Automatic Parallelization

- Example of compiled information output

```
(line-no.)(optimize)

                <<< Lo
                <<<  [PARALLELIZATION]
                <<<    Standard iteration count: 1000
                <<<  [OPTIMIZATION]
                <<<    SIMD(VL: 4)
                <<<    SOFTWARE PIPELINING
                <<< Loop-information  End >>>
     5  pp    8v    for (i = 0; i < N; ++i) {
     6  p     8v      a[ i ] = b[ i ] + b[ i+1 ];
     7  p     8v    }
                          :
```

Parallelization information on DO loops
- pp: Parallelized
- m: Includes both parallelized and not-parallelized parts
- S: Not parallelized
- Blank: Not parallelization target

Parallelization information on executable statements
- p: Parallelizable
- m: Includes both parallelizable and not-parallelizable parts
- s: Not parallelizable

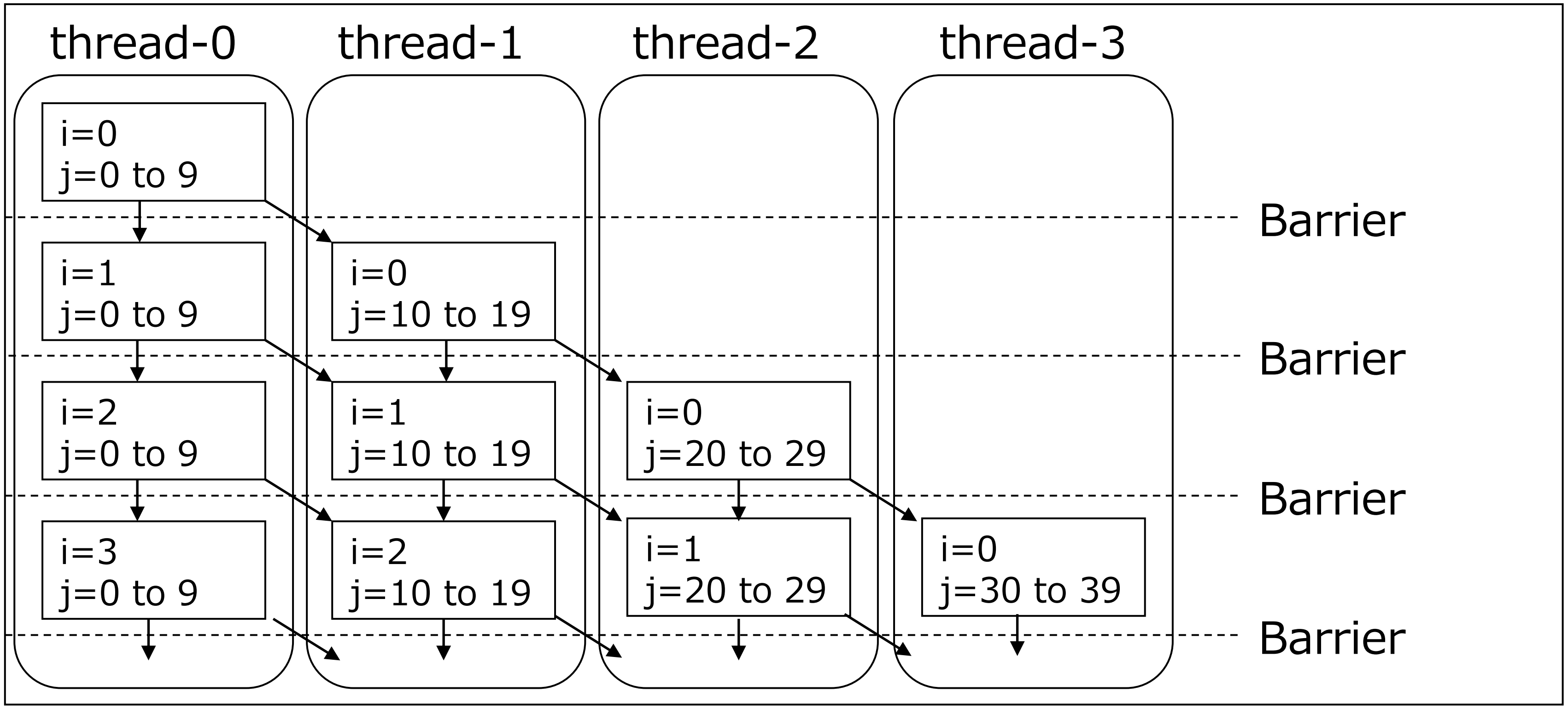
Pipeline Parallelism

- Pipeline parallelism

```
void sub() {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < 40; ++j) {
            a[ i ][ j ] = a[ i+1 ][ j ] + a[ i ][ j+1 ];
        }
    }
}
```

Pipelines have data dependency across cycles
-> Normal parallelization not practical

Pipelines do not have diagonal data dependency
-> Parallelization by diagonally slicing inner loops



Debug Functions for Fortran and C/C++ Trad Mode

- Built-in debug function
- Debug function for abnormal termination
- Hook function

Built-in Debug Function (1/2)

- You can use the following built-in debug function with the C/C++ compiler.

Compiler Option	-Nquickdbg
Test Item	subchk heapchk
Output Information	- Error message - Line number where error occurred - Variable name
Thread Parallelization Support	OpenMP and automatic parallelization supported

Built-in Debug Function (2/2)

- Usage
  Specify the following compiler option.

```
-Nquickdbg [=subchk | heapchk | inf_detail | inf_simple]
```

-
  ```
  $ fccpx  -Nquickdbg  test.c
  ```
  - If not specified, the subparameters assume the following values:
    –Nquickdbg=subchk –Nquickdbg=heapchk –Nquickdbg=inf_detail

- Arguments (subparameters) for testing

Argument	Test Description
subchk	Checks of the range when referencing arrays
heapchk	Checks of memory release and out-of-bounds write

- Arguments (subparameters) for displaying diagnostic messages

Argument	Displayed Content
inf_detail	Error message, line number, variable name
inf_simple	Error message, line number ＊Reduced impact on execution performance＊

Debug Function for Abnormal Termination (1/2)

- If the program is terminated abnormally, you can output information from the execution time to help in investigating the cause.

Compiler Option		-NRtrap
Caught Signals		SIGILL(04): Incorrect instruction executed SIGFPE(08): Arithmetic exception SIGBUS(10): Storage protection exception SIGSEGV(11): Segmentation exception SIGXCPU(30): CPU time interrupt
Output Information	General Abend	• Signal number • Signal code for the abend cause • Detailed information of signal code
	SIGXCPU	• Message

Debug Function for Abnormal Termination (2/2)

- Instructions
  Specify the following compiler option.

```
-NRtrap
```

```
$ fccpx -NRtrap  test.c
```

- Output information

Signal No.	Output Message
SIGILL SIGBUS SIGSEGV	jwe0019i-u The program was terminated abnormally with signal number *SSSSSSS*.signal identifier = *NNNNNNNNNN,(Detailed information.)* *SSSSSSS*: SIGILL, SIGBUS, or SIGSEGV *NNNNNNNNNN*: Signal code for the abend cause *(Detailed information.)*: Detailed information of signal code
SIGXCPU	jwe0017i-u The program was terminated with signal number SIGXCPU.

Hook Function (1/2)

- You can use a user-defined function to check program behavior by calling the function from a specific point in the program.

Compiler Option	-Nhook_func
User-Defined Function Name	user_defined_proc
User-Defined Function Arguments	FLAG: Information on user-defined function call source NAME: Function name at call source LINE: Line number at call source THREAD: Thread identification number (for OpenMP/automatic parallelization)
User-Defined Function Call Location	• Program entry/exit point • Function entry/exit point If -Kopenmp or -Kparallel is enabled, you can call the function from the following location in addition to above: • Parallel region (OpenMP/automatic parallelization) entry/exit point

Hook Function (2/2)

- Instructions
  Specify the following compiler option.

```
-Nhook_func
```

```
$ fccpx  -Nhook_func  test.c
```

- User-defined function format
  - Format

```
#include "fjhook.h"
void user_defined_proc(int *FLAG, char *NAME, int *LINE, int *THREAD)
```

  - Arguments

FLAG: Indicates the user-defined function call source.

0: Program entry point	1: Program exit point	2: Function entry point	3: Function exit point
4: Parallel region entry point		5: Parallel region exit point	
6 to 99: System reserved		100: Available to users	

NAME: Indicates the function name at the call source.
The argument can be referenced only when FLAG is 2, 3, 4, 5, or 100 or higher.
LINE: Indicates the line number at the call source.
The argument can be referenced only when FLAG is 2, 3, 4, 5, or 100 or higher.
THREAD: Indicates the identification number of the thread calling the user-defined function.
(OpenMP/automatic parallelization)
The argument can be referenced only when FLAG is 2, 3, 4, 5, or 100 or higher.

Large Page

About Large Page

- What is a large page?
  - Allocating memory (large page) with a larger page size than a normal page to applications handling data on a larger scale has the following effects:
    - Reduces the overhead incurred by the CPU address translation process
    - Improves memory access performance

  - In the A64FX system environment, the normal page size is 64 KiB, and the size available as a large page is 2 MiB.
    - You can set the following operations by setting environment variables:
      - Enabling/Disabling the large page allocation operation
      - Enabling/Disabling the large page allocation operation for the stack area
      - Selecting the paging method (page allocation trigger) for each memory area
    - The various page sizes that can be used with McKernel 32 MiB, 1 GiB, 16 GiB and etc.

*For details, see Chapter 3 Large Page Library in the "Job Operation Software End-user's Guide for HPC Extensions".

Large Page Specifications

- Large page specifications

Memory Area	MP10/FX10/FX100	A64FX			
	Page Size	Page Size			Paging (Default is Underlined)
		Normal Page	Large Page Base	Large Page Base+Stack (Default)	
Text (.text)	8 KiB	64 KiB	64 KiB	64 KiB	-
Static data (.data)	4 MiB (default), 8 KiB, 32 MiB, 256 MiB	64 KiB	2 MiB	2 MiB	Always prepage
Static data (.bss)		64 KiB	2 MiB	2 MiB	demand | prepage
Stack (*1)		64 KiB	64 KiB	2 MiB	demand | prepage
Dynamic memory (*2)		64 KiB	2 MiB	2 MiB	demand | prepage
Shared memory		64 KiB	64 KiB	64 KiB	-

*1 This covers the process stack/main thread stack/thread stack area.
*2 This covers the process heap/main thread heap/thread heap/mmap area.

Environment Variables for Large Page Settings (1/2)

● Basic settings/Paging method settings

Environment Variable Name	Specified Value (Default is Underlined)	Description
XOS_MMM_L_HPAGE_TYPE	<u>hugetlbfs</u> | none	With this setting, select whether to enable or disable the operation of large page allocation using the large page library. "hugetlbfs" enables large pages using HugeTLBfs. "none" does not enable large pages using the large page library.
XOS_MMM_L_LPG_MODE	<u>base+stack</u> | base	With this setting, select whether to enable or disable the operation of large page allocation for the stack region and thread stack region. "base+stack" enables large pages not only for the static data and Dynamically allocated memory region but also for the stack region and thread stack region. "base" enables large pages only for the static data and dynamic memory storage region. Large pages are not enabled for the stack region and thread stack region.
XOS_MMM_L_PAGING_POLICY	[demand | <u>prepage</u>]: [<u>demand</u> | prepage]: [demand | <u>prepage</u>]	With this setting, select the paging method (page allocation trigger) for each memory region. "demand" represents the demand paging method, and "prepage" represents the prepaging method. This variable specifies the paging method for three memory region delimited by a colon (:). The first specified method applies to the .bss region of static region. (This specified paging method does not cover the .data region of static data. "prepage" is always the value for this region.) The second specified method applies to the stack region and thread stack region. The third specified method applies to Dynamically allocated memory region. If a value not in the Specified Value column is specified, the respective value in "prepage:demand:prepage" is assumed specified.

Environment Variables for Large Page Settings (2/2)

● Settings for tuning (Environment variables specific to large pages)

Environment Variable Name	Specified Value (Default is Underlined)	Description
XOS_MMM_L_ARENA_FREE	<u>1</u> | 2	This setting relates to how to handle the heap area released by free(3). Specify "1" to immediately release the memory that can be released. Specify "2" to never release memory but instead pool all memory for reuse.
XOS_MMM_L_ARENA_LOCK_TYPE	0 | <u>1</u>	This setting relates to the memory allocation policy. "0" means memory acquisition performance has priority. "1" means memory utilization efficiency has priority.
XOS_MMM_L_MAX_ARENA_NUM	Integer value between <u>1</u> and INT_MAX [decimal number]	Set the number of arenas that can be generated (total for the process heap area and thread heap area). This setting is enabled when XOS_MMM_L_ARENA_LOCK_TYPE=0.
XOS_MMM_L_HEAP_SIZE_MB	Integer value between <u>MALLOC_MMAP_THRESHOLD x 2</u> and ULONG_MAX <MiB> [decimal number]	Set the size of memory acquired when generating and extending the thread heap area in order to use the thread heap area.
XOS_MMM_L_COLORING	0 | <u>1</u>	With this setting, enable or disable cache coloring. Cache coloring mitigates L1 cache conflicts of the processor. "0" does not enable cache coloring. "1" enables cache coloring when the size of memory being acquired by mmap(2) is equal to or greater than MALLOC_MMAP_THRESHOLD_ (default: 128 MiB).
XOS_MMM_L_FORCE_MMAP_THRESHOLD	<u>0</u> | 1	Set whether to prioritize mmap(2) when the size of memory being acquired is equal to or greater than MALLOC_MMAP_THRESHOLD_ (default: 128 MiB). "0" does not prioritize mmap(2). First, a search for free memory in the heap area returns any free memory found in the area. Then, mmap(2) acquires memory only when no free memory has been found in the heap area. "1" prioritizes mmap(2). Without a search for free memory in the heap area, mmap(2) acquires memory (in spite of any free memory).

■ For details on environment variables (MALLOC_MMAP_THRESHOLD_, etc.) for glibc , see the user's guide.

Precautions (Side Effects) Related to Memory Usage

- Static data (.data) area

  Both side effects 1 and 2 always occur.

- Static data (.bss) area

  If the following conditions are met, side effects 1 and 2 occur.

  a. The .dynsym section (dynamic section) has a symbol.
  b. The symbol has an address within the range of the bss area (bss_start, bss_end) of the main program.
  c. The symbol is global (STB_GLOBAL) or weak (STB_WEAK).
  d. The symbol type is variable (STT_OBJECT).
  e. The size of the symbol is larger than 0.

- Side effects

  1. A large page may use double or triple the memory area.
  2. The prepaging method ("prepage") is the active method even when the demand paging method ("demand") is set for the static data (.bss) area in XOS_MMM_L_PAGING_POLICY.

Paging Policy of Large Page

Since data comes from CMG0 in prepaging, performance cannot reach that of 48-thread streams. With the method changed to demand paging, data is put on the running CMG, and performance is significantly higher.

Source

```
14                  Subroutine sub(n,iter,x1,x2,y1)
15                    real(8) :: x1(n), x2(n), y1(n),c0
16                    integer n,i,k
17                    c0=2.0
18
19                    call fapp_start("sub",0,0)
20     1               do k=1,iter
21     1             !$omp parallel do
               <<< Loop-information Start >>>
               <<<  [OPTIMIZATION]
               <<<    SIMD(VL: 8)
               <<<    SOFTWARE PIPELINING(IPC: 2.45, ITR:
128, MVE: 2, POL: S)
               <<<    PREFETCH(SOFT) : 10
               <<<     SEQUENTIAL : 10
               <<<      x2: 4, x1: 4, y1: 2
               <<<    ZFILL         :
               <<<      y1
               <<< Loop-information  End >>>
22     2   p   v        do i=1,n
23     2   p   v          y1(i) = x1(i) + c0 * x2(i)
24     2   p   v        end do
25     1               enddo
 :                     ......
30                    parameter(N=45000000,ITER=100)
31                    real*8 x1(N),x2(N),y1(N)
32                    call init(N,ITER,x1,x2,y1)
33                    call sub(N,ITER,x1,x2,y1)
```

Stream (Data size: About 1 GB)

	Memory throughput (GB/s)
prepage (default)	93 GB/s
demand	804 GB/s

Compiler option: -Kfast,openmp
-Kprefetch_sequential=soft -Kprefetch_line=9
-Kprefetch_line_L2=70 -Kzfill=18

Lock Type of Large Page

malloc performance is higher when XOS_MMM_L_ARENA_LOCK_TYPE=0 is specified. (Reduced execution time from 0.56 seconds to 0.35 seconds, a performance increase of 1.60 times)

Source

```
1                 subroutine sub(n,m,iter,x1,x2,y2)
2                   integer(8) :: pZ1(iter)
3                   real(8) :: x1(n), x2(n), y2(n,m),c0
4                   c0=2.0
5
6                 !$omp parallel do shared(n,m,iter,x1,x2,c0,y2) private(pZ1,i,j,k) default(none)
               <<< Loop-information Start >>>
               <<<  [OPTIMIZATION]
               <<<    PREFETCH(HARD) Expected by compiler :
               <<<      x1, x2, y2
               <<< Loop-information  End >>>
7     1   p              do k=1,m
               <<< Loop-information Start >>>
               <<<  [OPTIMIZATION]
               <<<    PREFETCH(HARD) Expected by compiler :
               <<<      (unknown)
               <<< Loop-information  End >>>
8     2   p   s            do j=1,iter
9     2   p   m              pZ1(j) = malloc(8 * n)
10    2   p   v            end do
11    1
               <<< Loop-information Start >>>
               <<<  [OPTIMIZATION]
               <<<    SIMD(VL: 8)
               <<<    SOFTWARE PIPELINING(IPC: 3.50, ITR: 144, MVE: 4, POL: S)
               <<<    PREFETCH(HARD) Expected by compiler :
               <<<      x1, x2, y2
               <<< Loop-information  End >>>
12    2   p  2v            do i=1,n
13    2   p  2v              y2(i,k) = x1(i) + c0 * x2(i)
14    2   p  2v            end do
15    1
16    2   p   s            do j=1,iter
17    2   p   s              call free( pZ1(j))
18    2   p   s            end do
19    1   p              end do
20                end subroutine sub
21
22                program main
23                  parameter(N=1048512,ITER=80)
24                  real*8 x1(N),x2(N),y2(N,12)
25                  call sub(N,12,ITER,x1,x2,y2)
26                end program main
```

Behavior of Large Page Shortage (Fugaku only)

- XOS_MMM_L_HUGETLB_FALLBACK
  - This setting specifies the behavior of large page shortage during malloc.
  - [1] causes try allocating memory shortage with normal pages.
    If a normal page is unable to acquire memory to satisfy a malloc request, the process ends with out of memory. This setting is effective for programs that operate for normal page specification (XOS_MMM_L_HPAGE_TYPE=none) but fail to execute due to memory exhaustion for large page specification. By enabling this setting, it is possible to make the program work by using large pages as much as possible while compensating for the shortage with normal pages and increasing the amount of memory that can be obtained with malloc.
  - [0] causes the process to terminate by out of memory when a large page shortage occurs. (Default)
  - Note 1: The following settings are required to enable this setting.
    I. XOS_MMM_L_HPAGE_TYPE=hugetlbfs and
    II. XOS_MMM_L_PAGING_POLICY=any:any:prepage and
    III. XOS_MMM_L_ARENA_LOCK_TYPE=1 and
    IV. XOS_MMM_L_MAX_ARENA_NUM=1

    These four conditions are the default settings for the large page library and do not need to be explicitly specified unless otherwise specified.
  - Note 2: When this setting is enabled (Specify 1), a mixed malloc area of normal page and large page is generated, but when this area is used as a communication buffer of Tofu, the whole area is managed as normal page on Tofu. Therefore, the communication performance when used as a communication buffer cannot be expected to be improved from the case of normal page specification.

Timers Supported by Fortran

- Timer Specifications
- Timer Precision

Timer Specifications (1/3)

- Specifications of the major timer routines

No.	Routine Name	Function	Format	What is Measured	Unit Returned by Routine
1	DATE_AND_TIMEb uilt-in subroutine	Obtains the date and time.	CALL DATE_AND_TIME ([DATE , TIME , ZONE , VALUES]) DATE: Sets the current date in the CCYYMMDD format. (CC: century, YY: year, MM: month, DD: day) TIME: Sets the current time in the hhmmss.sss format. (hh: hour, mm: minute, ss.sss: second) ZONE: Sets the time zone offset from UTC in the shhmm format. (s: sign, hh: hour, mm: minute) VALUES(1): Year VALUES(2): Month VALUES(3): Day VALUES(4): Time zone offset from UTC in minutes VALUES(5): Hour VALUES(6): Minute VALUES(7): Second VALUES(8): Millisecond	Current date/time	
2	GETTIM service subroutine	Obtains the current time.	CALL GETTIM (hour , minute , second , second1_100) hour: Sets the current hour. minute: Sets the current minute. second: Sets the current second. second1_100: Sets the current hundredth of a second.	Current time	
3	FDATE service subroutine	Obtains the current date and time by converting them to ASCII code.	CALL FDATE (string) string: Sets the current date and time in the order of day of the week, month, day, time, and year.	Current date and time	
4	ITIME service subroutine	Obtains the current hour, minute, and second.	CALL ITIME (ia) ia(1): Sets the current hour. ia(2): Sets the current minute. ia(3): Sets the current second.	Current hour, minute, and second	
5	GETTOD service subroutine	Obtains the current real time. The real time is the time in microseconds elapsed since a specific time point in the past. Usually, it represents the time since system boot.	CALL GETTOD (g) g: Sets the elapsed time in microseconds.	Wall clock time	Microsecond

Timer Specifications (2/3)

- Specifications of the major timer routines

No.	Routine Name	Function	Format	What is Measured	Unit Returned by Routine
6	GMTIME service subroutine	Obtains the specified system clock time information to show the second, minute, hour, day, month, year, day of the week, total number of days since January 1, and observation of daylight saving time according to Greenwich Mean Time.	CALL GMTIME (time , t) time: Specifies the system clock time. The following array is set from the system clock time with the specified "time" value according to Greenwich Mean Time: t(1): Second t(2): Minute t(3): Hour t(4): Day t(5): Month t(6): Total number of years since 1990 t(7): Total number of days since Sunday t(8): Total number of days since January 1 t(9): The setting is 0 for standard time and 1 for daylight saving time.	Wall clock time	
7	LTIME service subroutine	Obtains the specified system clock time information to show the second, minute, hour, day, month, year, day of the week, total number of days since January 1, and observation of daylight saving time according to the local time.	CALL LTIME (time , t) time: Specifies the system clock time. The following array is set from the system clock time with the specified "time" value according to the local time: t(1): Second t(2): Minute t(3): Hour t(4): Day t(5): Month t(6): Total number of years since 1990 t(7): Total number of days since Sunday t(8): Total number of days since January 1 t(9): The setting is 0 for standard time and 1 for daylight saving time.	Wall clock time	
8	omp_get_wtime routine	Obtains the current real time. The real time is the time in seconds elapsed since a specific time point in the past. Usually, it represents the time since system boot.	y = omp_get_wtime ( ) y: Returns the elapsed time in seconds.	Wall clock time	Second
9	SECNDS service function	Obtains the number of seconds by subtracting the value specified in the first argument from the number of seconds elapsed since midnight in the system clock time.	y = SECNDS (sec) y: Returns the number of seconds obtained by subtracting the sec value from the number of seconds elapsed since midnight in the system clock time. sec: Specifies a value in seconds to subtract from the number of seconds elapsed since midnight in the system clock time.	Wall clock time	Second
10	SYSTEM_CLOCK built-in subroutine	Obtains the total time elapsed since midnight. The total time is an elapsed time in seconds. Usually, it represents the time since system boot.	CALL SYSTEM_CLOCK ([COUNT , COUNT_RATE , COUNT_MAX]) COUNT: Sets the time elapsed from midnight until the present time. COUNT_RATE: Sets the rate of counting (=1000) per second by the processing system. COUNT_MAX: Sets the maximum COUNT value (=86399999).	Wall clock time	Millisecond
11	RTC service function	Obtains the total number of seconds elapsed since 0:00 on January 1, 1970 in UTC.	y = RTC ( ) y: Sets the total number of seconds elapsed since 0:00 on January 1, 1970.	Wall clock time	Second

Timer Specifications (3/3)

- Specifications of the major timer routines

No.	Routine Name	Function	Format	What is Measured	Unit Returned by Routine
12	TIME service function	Obtains the time elapsed in seconds since 00:00:00 GMT (January 1, 1970).	iy = TIME ( ) iy: Returns the elapsed time in seconds since 00:00:00 GMT (January 1, 1970).	Wall clock time	Second
13	TIMEF service function	Returns the elapsed time since the last called TIMEF service function.	y = TIMEF ( ) y: Returns the elapsed time since the last executed TIMEF service function.	Wall clock time	Second
14	TIMER service subroutine	Obtains the time elapsed in hundredths of seconds since midnight.	CALL TIMER(ix) ix: Sets the time elapsed in hundredths of seconds since midnight.	Wall clock time	Second
15	CLOCK service subroutine	Obtains the CPU time elapsed since the execution start of an executable program. The CPU time represents the time used by the processes executing the program and all their internal threads.	CALL CLOCK (g , i1 , i2) g: Sets the CPU time in the unit specified in i1. i1: Specifies the unit (second, millisecond, microsecond) for the return value. i2: Specifies the type of the variable specified in g.	CPU time used by current processes and their internal threads	Any of following units depending on specification: - Second - Millisecond - Microsecond
16	CLOCKM service subroutine	Obtains the CPU time elapsed since the execution start of an executable program. The CPU time represents the time used by the processes executing the program and all their internal threads.	CALL CLOCKM (i) i: Sets the CPU time in milliseconds.	CPU time used by current processes and their internal threads	Millisecond
17	CLOCKV service subroutine	Obtains the CPU time elapsed since the execution start of an executable program. The CPU time represents the time used by the processes executing the program and all their internal threads. The routine is compatible with vector machines.	CALL CLOCKV (g1 , g2 , i1 , i2) g1: Always sets 0. * VU time is set in vector machines. g2: Sets the CPU time in the unit specified in i1. i1: Specifies the unit (second, millisecond, microsecond) for the return value. i2: Specifies the type of the variable specified in g2.	CPU time used by current processes and their internal threads	Any of following units depending on specification: - Second - Millisecond - Microsecond
18	CPU_TIME built-in subroutine	Obtains the CPU time elapsed since the execution start of an executable program. The CPU time represents the time used by the processes executing the program and all their internal threads.	CALL CPU_TIME (TIME) TIME: Sets the CPU time in seconds.	CPU time used by current processes and their internal threads	Second
19	DTIME service function	Obtains the CPU time from the last called DTIME service function. The CPU time represents the time used by the processes executing the program and all their internal threads.	y = DTIME (tm) y: Returns the CPU time since the last called DTIME service function. tm(1): Sets the user CPU time in seconds. tm(2): Sets the system CPU time in seconds.	CPU time used by current processes and their internal threads	Second
20	ETIME service function	Obtains the CPU time elapsed since the execution start of an executable program. The CPU time represents the time used by the processes executing the program and all their internal threads.	y = ETIME (tm) y: Returns the CPU time elapsed since the execution start of an executable program. tm(1): Sets the user CPU time in seconds. tm(2): Sets the system CPU time in seconds.	CPU time used by current processes and their internal threads	Second
21	SECOND service function	Obtains the user CPU time elapsed since the execution start of an executable program. The CPU time represents the time used by the processes executing the program and all their internal threads.	y = SECOND ( ) y: Returns the user CPU time in seconds elapsed since the execution start of an executable program.	CPU time used by current processes and their internal threads	Second

Timer Precision (1/2)

- Precision of the major timer routines (Timer overhead)

No.	Routine Name	Precision Resolution	Overhead (μs)	Thread Safe	Implementation	GCC Overhead (μs)	Fujitsu/GCC	Remarks
1	DATE_AND_TIME built-in subroutine	1/1,000,000	77.37 (Improved version) 47.27	✓	gettimeofday localtime	163.13	0.47	
2	GETTIM service subroutine	1/1,000,000	38.37	✓	gettimeofday localtime			
3	FDATE service subroutine	1/1,000,000	18.58	✓	time ctime_r	78.71	0.24	
4	ITIME service subroutine	1/1,000,000	36.07	✓	time localtime	69.27	0.52	
5	GETTOD service subroutine	1/100,000,000	0.02	✓	arm asm instruction time stamp counter gmtime_r localtime_r			Resolution: Value of (1/cntfrq_el0)
6	GMTIME service subroutine		5.72	✓	gmtime_r	58.21	0.10	
7	LTIME service subroutine		10.81	✓	localtime_r	67.59	0.16	
8	omp_get_wtime routine	FJOMP: 1/100,000,000 libomp: 1/1,000,000	1.00	✓	FJOMP: arm asm instruction time stamp counter libomp: gettimeofday localtime	6.05	0.17	Resolution: Value of (1/cntfrq_el0)
9	SECNDS service function	1/1,000,000	49.05	✓	gettimeofday localtime	77.31	0.63	
10	SYSTEM_CLOCK built-in subroutine	1/100,000,000	20.98 (Improved version) 1.42	✓	arm asm instruction time stamp counter	5.23	4.01 (Improved version) 0.27	Resolution: Value of (1/cntfrq_el0)
11	RTC service function	1/1,000,000	5.21	✓	time			

Timer Precision (2/2)

- Precision of the major timer routines (Timer overhead)

No.	Routine Name	Precision Resolution	Overhead (µs)	Thread Safe	Implementation	GCC Overhead (µs)	Fujitsu/GCC	Remarks
12	TIME service function	1/1,000,000	5.28	✓	time	5.03	1.05	
13	TIMEF service function	1/1,000,000	7.26	✓	time			
14	TIMER service subroutine	1/1,000,000	49.85	✓	gettimeofday localtime			
15	CLOCK service subroutine	1/1,000,000	12.82	✓	getrusage			
16	CLOCKM service subroutine	1/1,000,000	13.70	✓	getrusage			
17	CLOCKV service subroutine	1/1,000,000	14.12	✓	getrusage			
18	CPU_TIME built-in subroutine	1/1,000,000	18.09	✓	getrusage	5.39	3.36	
19	DTIME service function	1/1,000,000	14.12	✓	getrusage	10.20	1.38	
20	ETIME service function	1/1,000,000	13.33	✓	getrusage	9.37	1.42	
21	SECOND service function	1/1,000,000	14.27	✓	getrusage	5.91	2.41	

Main Entry Names in Compiler Runtime Library

- Main Entry Names in Compiler Runtime Library

Main Entry Names in Compiler Runtime Library

Entry Name	Functional Outline
__jwe_c	Module called from the compiler (check routine, Fortran multi-phase processing, etc.)
__jwe_ca	COARRAY
__jwe_d	Intrinsic debugging, quick debugging
__jwe_e	Error processing of the runtime library
__jwe_f	Internal routine such as for operations and type conversion called from inside the runtime library
__jwe_hook	HOOK function
__jwe_i	Fortran IO processing
__jwe_o	Parallel processing (mainly, OpenMP)
__jwe_p	Parallel processing (parallel control, automatic parallelization)
__jwe_s	Processing module of the Fortran service routine
__jwe_t	Runtime information output function
__jwe_x	Runtime library control (initialization, end, exception, area management, etc.)
__mpc__	C/C++ (trad mode) OpenMP
__f_	Fortran function (transformational function, array function, query function, etc.)
__g_	Floating-point math operation-related function
__plvla	mfunc=1
__vm_	mfunc=2
__v_	mfunc=3
omp_	OpenMP routine
kmp_	LLVM OpenMP

Variations in Elapsed Time of Operation Region

Causes of Variations in Elapsed Time of Operation Region

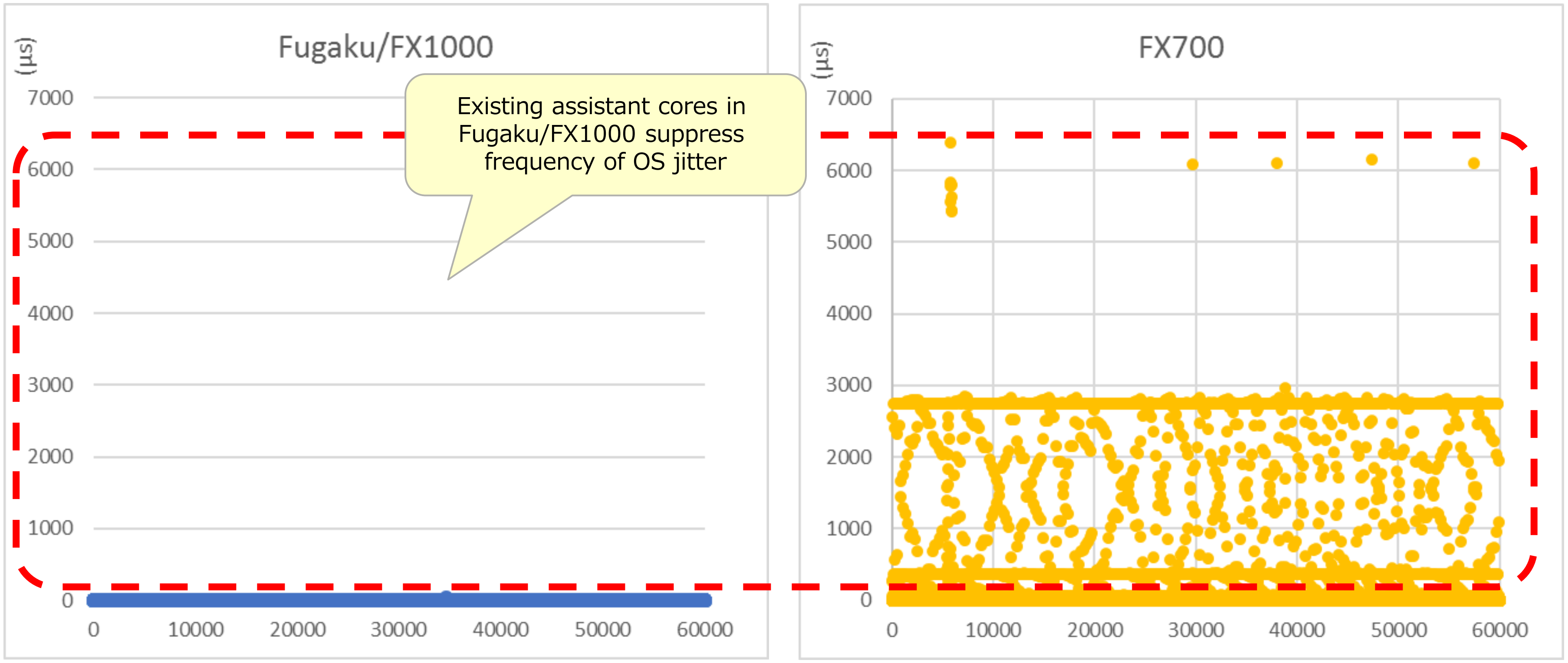
FUJITSU

- A variation in the elapsed time may occur in some cases, even for the same arithmetic processing. The potential causes are as follows:
  - (1) Latency due to node OS jitter
  - (2) Use of other CMG memory
    ->The memory used may span across CMGs.
    This is primarily due to the use of dynamic libraries.
  - (3) Difference in the accessed array address (physical address)
  - (4) Execution timing fluctuations due to an OoO operation
    -> Bias of operation pipeline a or b, load and store pipeline control, etc.
  - (5) Differences in L2 cache access latency
  - (6) Performance impact due to a directory path name change

(1), (5), and (6) are explained on the subsequent pages.

Effects of Node OS Jitter

- The Fugaku/FX1000 contain assistant cores in the CPUs.
  The result is fewer OS jitter than in the FX700.
- The following graphs show the aggregate results of executing FWQ using 48 cores (48 threads) in the node.



Latency Due to Differences in L2 Cache Access Latency

- **Acquisition of L1D latency**

  After execution on all the cores in the CPU (12 threads x 4 MPI execution), performance events were acquired using the fapp command. The average latency of the L1D cache miss processing of each core was calculated from the following formula.

- **Latency formula**

  Average latency of L1D cache miss processing
  = L1_MISS_WAIT / L1D_CACHE_REFILL

  * Source: *A64FX Microarchitecture Manual*

- **Validation code**

```
39                     !$omp parallel
                  <<< Loop-information Start >>>
                  <<<  [OPTIMIZATION]
                  <<<    PREFETCH(HARD) Expected by compiler :
                  <<<      x2, x1, y
                  <<< Loop-information  End >>>
40    1                  DO j = 1, iter
41    1                !$omp do
                  <<< Loop-information Start >>>
                  <<<  [OPTIMIZATION]
                  <<<    SIMD(VL: 8)
                  <<<    SOFTWARE PIPELINING(IPC: 3.25, ITR: 144, MVE: 4,
POL: S)
                  <<<    PREFETCH(HARD) Expected by compiler :
                  <<<      x2, x1, y
                  <<< Loop-information  End >>>
42    2  p  2v              DO i = 1, n
43    2  p  2v                y(i)=x1(i) + c0 * x2(i)
44    2  p  2v              ENDDO
45    1                !$omp end do nowait
46    1                  ENDDO
47                     !$omp end parallel
```

iter=12000

- **Execution results**

Process 1 delayed in evaluation environment

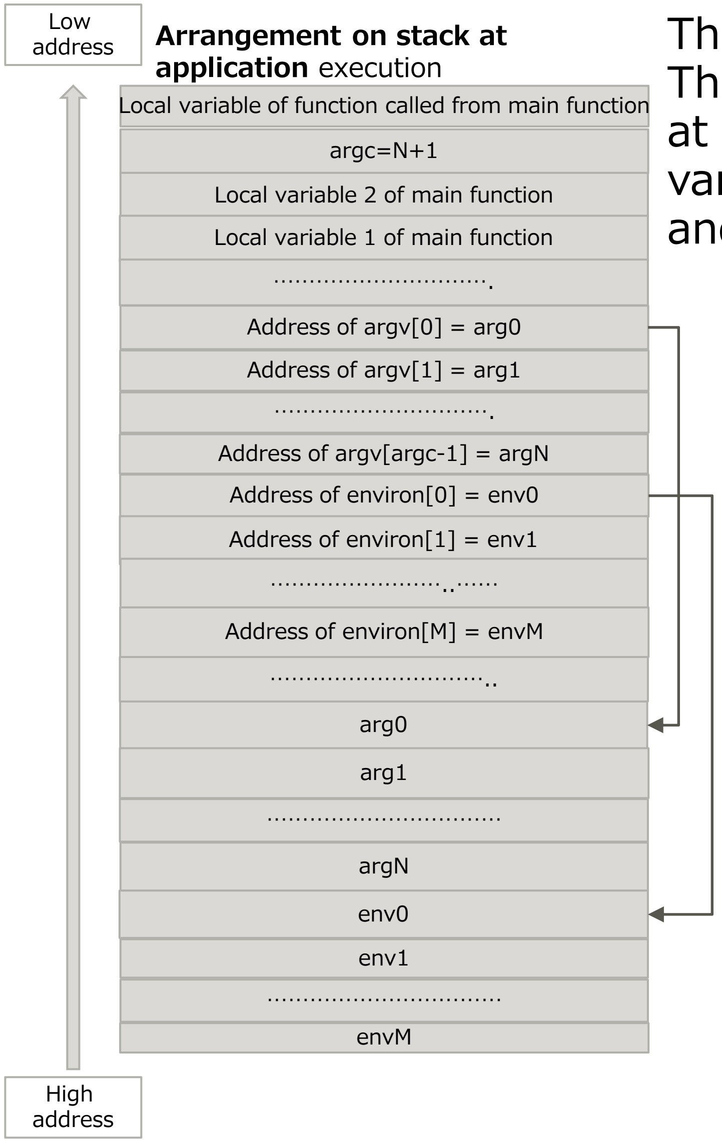
Process 0	Process 1
Core 12: 60.3	Core 24: 67.2
Core 13: 61.4	Core 25: 50.2
Core 14: 61.4	Core 26: 68.6
Core 15: 60.2	Core 27: 67.8
Core 16: 61.8	Core 28: 50.3
Core 17: 59.6	Core 29: 68.5
Core 18: 61.9	Core 30: 53.8
Core 19: 60.7	Core 31: 68.8
Core 20: 63.1	Core 32: 54.9
Core 21: 62.8	Core 33: 69.7
Core 22: 63.3	Core 34: 59.3
Core 23: 66.3	Core 35: 71.0

About 7% slower than slowest core in other processes

Process 2	Process 3
Core 36: 60.3	Core 48: 60.3
Core 37: 61.5	Core 49: 61.4
Core 38: 61.5	Core 50: 61.4
Core 39: 60.2	Core 51: 60.2
Core 40: 61.8	Core 52: 61.8
Core 41: 59.6	Core 53: 59.6
Core 42: 61.9	Core 54: 61.8
Core 43: 60.7	Core 55: 60.7
Core 44: 63.1	Core 56: 63.1
Core 45: 62.8	Core 57: 62.7
Core 46: 63.3	Core 58: 63.3
Core 47: 66.3	Core 59: 66.3

Apparent difference in latency between cores

Performance Impact Due to Directory Path Name Change (1/3)


Low address
Arrangement on stack at application execution
Local variable of function called from main function
argc=N+1
Local variable 2 of main function
Local variable 1 of main function
Address of argv[0] = arg0
Address of argv[1] = arg1
Address of argv[argc-1] = argN
Address of environ[0] = env0
Address of environ[1] = env1
Address of environ[M] = envM
arg0
arg1
argN
env0
env1
envM
High address

The stack is used from high addresses to low addresses. The figure on the left shows the arrangement on the stack at application execution. The order is environment variables (env1 to envM), then arguments (arg1 to argN), and then local variables.

● Impact on performance

Changing the directory path name for a load module location or adding an environment variable at load module execution can have an impact on performance during execution of the load module.

● Cause of performance irregularities

Changing a directory path name or adding an environment variable at load module execution may cause the location address of a local variable defined within a function (located in the stack region) to shift.

Performance Impact Due to Directory Path Name Change (2/3)

Causes of shifting location addresses on stack

The set environment variables are different for each type of shell. The following explanation uses /bin/bash for an example.
The following three environment variables are special:

PWD	**Name of the current working directory**
OLDPWD	**Name of the current working directory before a move by the** cd command
_	**Load module path name**

The shifting of location addresses for the above three environment variables has the following six causes.

1. A full or relative path name has been specified for the execution of a load module, and the path name is set as is for the "_" environment variable.
2. Only the load module file name has been specified for the execution of a load module, and the load module path name is complemented by the PATH environment variable. However, the full path name is set for the "_" environment variable.
3. When the cd command moves to a directory, the OLDPWD environment variable is set to the directory name before the move. The directory moved to by the cd command is set to the PWD environment variable.
4. For the above reasons, if the load module is specified with the full path name or if the full path name of the load module is complemented by the PATH environment variable, the addresses of arguments shift slightly. As a result, the addresses of local variables of functions called from the main function also shift.
5. Also, changing the length of the full path name will change the length of the full path name of the load module set in the "_" environment variable. In turn, the length of the string stored as an environment variable increases, and the addresses of local variables of functions called from the main function shift.
6. When the cd command moves from the current directory before load module execution, a special environment variable called "OLDPWD" is set. In turn, the addresses of local variables of functions called from the main function shift.

Performance Impact Due to Directory Path Name Change (3/3)

- Operation check results on actual device

(1) Load module placed directly under root (/)
(2) Load module placed directly under the root home directory (/root) (16-byte shift relative to (1))
(3) Load module placed in the load module storage directory of root (/root/bin). The full path name is complemented by the PATH environment variable. (16-byte shift relative to (1))
(4) Load module placed in the os directory (/root/os) directly under the root home directory (/root). The load module is executed with a relative path after a move to /root/os. (32-byte shift relative to (1))

(1) Load module placed directly under root (/)

```
# /hellomodule4 1 2 3 4
hello, function call arg=function call argument address: 0x0000ffffffffea68
hello, function call local world: 0x0000ffffffffea90
hello, main argc=5: 0x0000ffffffffeaec
hello, main initialized function local world: 0x0000ffffffffeaf0
hello, main not initialized local world: 0x0000ffffffffeb30
hello, /hellomodule4: 0x0000ffffffffef61
hello, 1: 0x0000ffffffffef6f
hello, 2: 0x0000ffffffffef71
hello, 3: 0x0000ffffffffef73
hello, 4: 0x0000ffffffffef75
.....................................
hello, _=/hellomodule4: 0x0000ffffffffffda
```

(2) Load module placed directly under the root home directory (/root)

```
# /root/hellomodule4 1 2 3 4
hello, function call arg=function call argument address: 0x0000ffffffffea58
hello, function call local world: 0x0000ffffffffea80
hello, main argc=5: 0x0000ffffffffeadc
hello, main initialized function local world: 0x0000ffffffffeae0
hello, main not initialized local world: 0x0000ffffffffeb20
hello, /root/hellomodule4: 0x0000ffffffffef52
hello, 1: 0x0000ffffffffef65
hello, 2: 0x0000ffffffffef67
hello, 3: 0x0000ffffffffef69
hello, 4: 0x0000ffffffffef6b
.....................................
hello, _=/root/hellomodule4: 0x0000ffffffffffd0
```

(3) Load module placed in the load module storage directory of root (/root/bin). The full path name is complemented by the PATH environment variable.

```
# hellomodule4 1 2 3 4
hello, function call arg=function call argument address: 0x0000ffffffffea58
hello, function call local world: 0x0000ffffffffea80
hello, main argc=5: 0x0000ffffffffeadc
hello, main initialized function local world: 0x0000ffffffffeae0
hello, main not initialized local world: 0x0000ffffffffeb20
hello, hellomodule4: 0x0000ffffffffef50
hello, 1: 0x0000ffffffffef5d
hello, 2: 0x0000ffffffffef5f
hello, 3: 0x0000ffffffffef61
hello, 4: 0x0000ffffffffef63
.....................................
hello, _=/root/bin/hellomodule4: 0x0000ffffffffffc8
```

(4) Load module placed in the os directory (/root/os) directly under the root home directory (/root). The load module is executed with a relative path after a move to /root/os.

```
# cd os
# ./hellomodule4 1 2 3 4
hello, function call arg=function call argument address: 0x0000ffffffffea48
hello, function call local world: 0x0000ffffffffea70
hello, main argc=5: 0x0000ffffffffeacc
hello, main initialized function local world: 0x0000ffffffffead0
hello, main not initialized local world: 0x0000ffffffffeb10
hello, ./hellomodule4: 0x0000ffffffffef4e
hello, 1: 0x0000ffffffffef5d
hello, 2: 0x0000ffffffffef5f
hello, 3: 0x0000ffffffffef61
hello, 4: 0x0000ffffffffef63
,.....................................
hello, _=./hellomodule4: 0x0000ffffffffffcb
hello, OLDPWD=/root: 0x0000ffffffffffdc
```

Revision History

Revision History

Version	Date	Details
1.1	May 14, 2020	- First published
1.2	Sep 30, 2020	- Correcting typographical errors and expressions by reviewing articles
1.3	Mar 31, 2021	- Correcting typographical errors and expressions by reviewing articles
1.4	Aug. 2021	- Fixing differences by increasing the number of software versions, and correcting typographical errors and expressions by reviewing articles - Added "Behavior of Large Page Shortage(Fugaku only)" page - Added shared libraries information for "LLVM OpenMP Library and Fujitsu OpenMP Library (2/2)" page
1.5	Jul. 2022	- Correcting typographical errors and expressions by reviewing articles
1.6	Mar. 2023	- Correcting typographical errors and expressions by reviewing articles