Low/Adaptive Precision Computation in ICCG solver for ill-conditioned problem

Masatoshi Kawai

Nagoya University, ITC

International Workshop on “Integration of Simulation/Data/Learning and Beyond”
45thASE Seminar (Advanced Supercomputing Environment)
Nov 29, 2023, Kashiwa, Japan & Online

Outline

1. Objective
2. Low/Adaptive precisions
3. Storage format
4. Numerical evaluations
5. Conclusion

Objective

Considering the effectiveness of low/adaptive precision on ICCG method.

Background

The effectiveness of the low/adaptive precisions are discussed in the field of deep learning, mainly.

If targeted data can be expressed in lower precision

↓

Use of lower precision reduces execution time
Because of improving an effectiveness of a SIMDization or reducing amount of memory transfer.

As same as practical simulations,

- The use of lower precision reduces the execution time.
- FP21 (adaptive precision) is evaluated on the seismic simulation on a GPU*1.

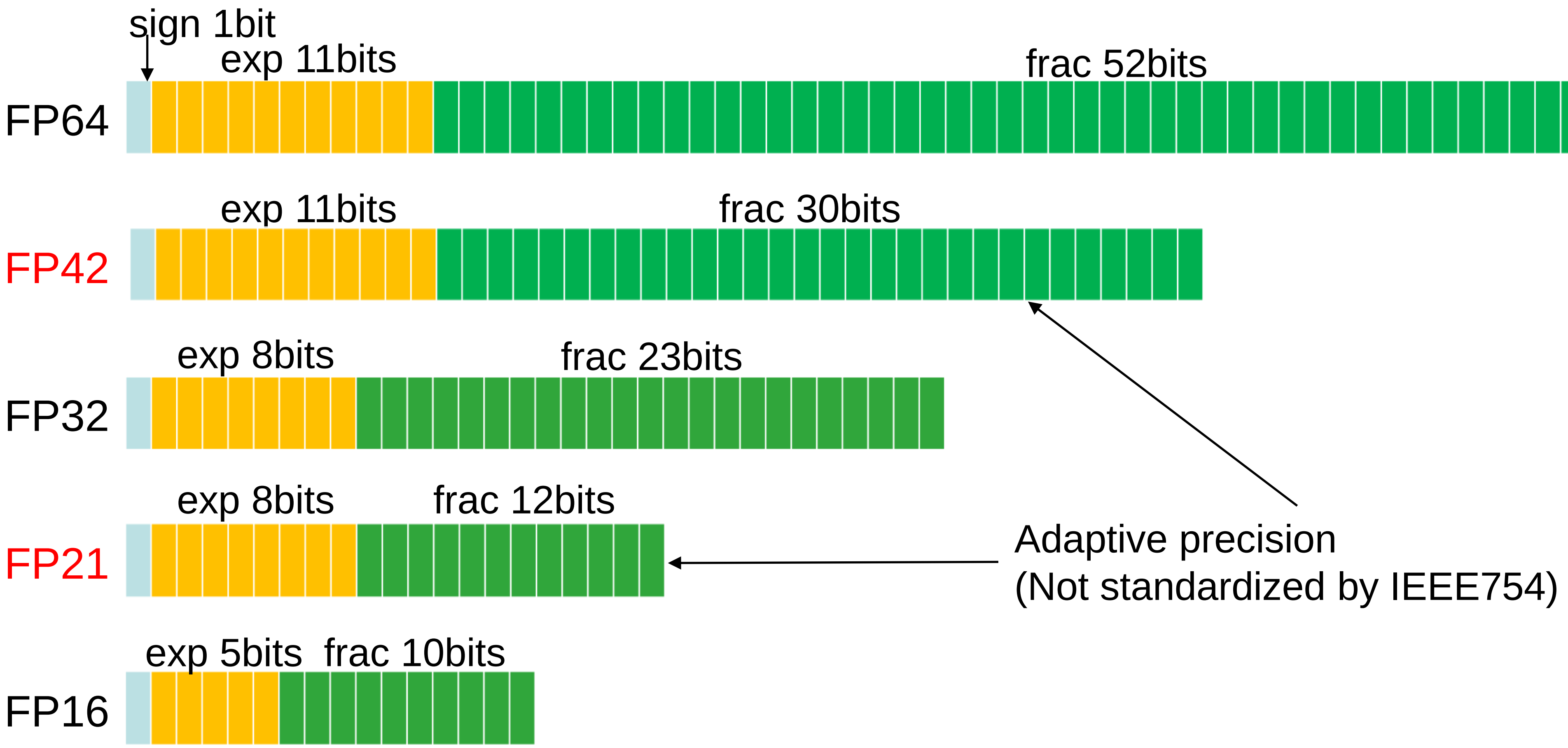
In this study, we evaluate the effectiveness of low/adaptive precision with iterative method on CPUs.

- ICCG is one of the most famous iterative method which require high accuracy of computations.
- The performance of the ICCG method is determined by memory bandwidth.

*1 T. Ichimura et al., "A Fast Scalable Implicit Solver for Nonlinear Time-Evolution Earthquake City Problem on Low-Ordered Unstructured Finite Elements with Artificial Intelligence and Transprecision Computing," SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, 2018, pp. 627-637

Data formats

Considering following data formats



Use FP21 and FP42 reduces data transfer between memory and CPU to 2/3 compared with FP32 and FP64.
For computing FP21 and FP42, it require data casting because of unsupported by FPUs.

Expressive ability of each data format

Wider data format have a higher expressive ability

It has strong impact on exponent part, especially.

Expressive ability translated to a decimal number

Formats	Significand : Number of decimal digits	Exponent : Maximum exponent in decimal
FP64	15.95	308
FP42	9.33	308
FP32	7.22	38
FP21	3.91	38
FP16	3.31	5

Expressive ability of the significand is computed as following

$$10^{y} = 2^{x+1}$$

$x+1$ is produced by hidden bit

$$y = (x+1)\log_{10} 2$$

Then, y denotes number of decimal digits, and x denotes number of bits of exponent part

Type casting between FP21 and FP32

FP32→FP21

```
#define fp21x3 integer(4)

function fp32x3_to_fp21x3_f(a1, a2, a3) result(b)
  implicit none
  real(4), intent(in) :: a1, a2, a3
  fp21x3 :: b
  fp21x3 c
  call cast_fp32_to_fp21x3(a1, c)
  b(1) = shiftr(iand(c, int(Z'fffff800', 4)), 11)
  call cast_fp32_to_fp21x3(a2, c)
  c = iand(c, int(Z'fffff800', 4))
  b(1) = ior(b(1), shiftl(c, 10))
  b(2) = shiftr(c, 22)
  call cast_fp32_to_fp21x3(a3, c)
  b(2) = ior(b(2), iand(c, int(Z'fffff800', 4)))
end function fp32x3_to_fp21x3_f

subroutine cast_fp32_to_fp21x3(a, b)
  implicit none
  fp21x3, intent(in)  :: a
  fp21x3, intent(out) :: b
  b = a
end subroutine cast_fp32_to_fp21x3
```

Left shows a Fortran pseudo code for type casting from FP21 to FP32

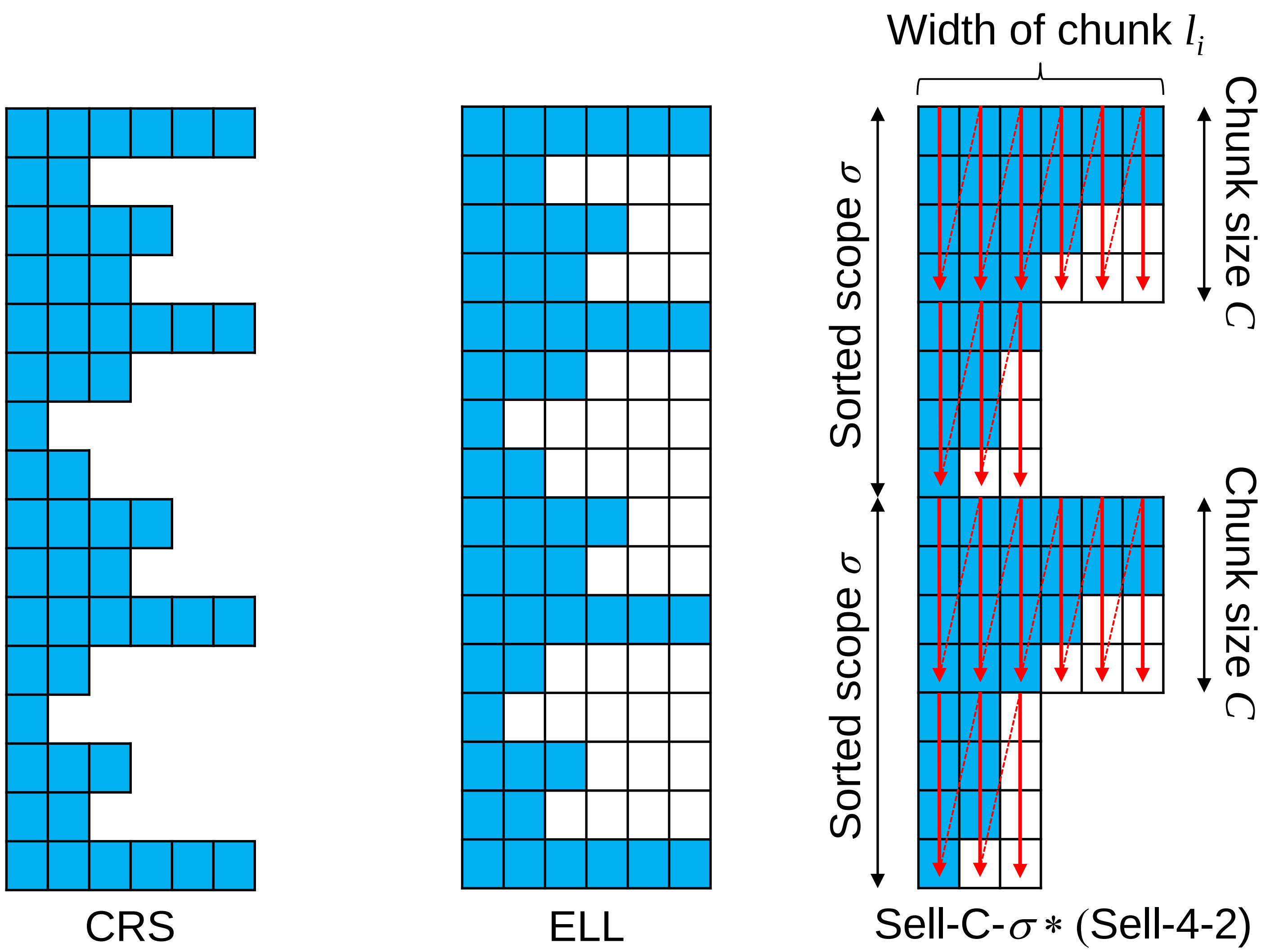
Three FP21 data are stored by two 32bits integer data format.

- We implement type casting without changing internal bit information (reinterpret cast) by calling subroutine with different argument data type.
- To SIMDize type casting calls, we add a link time optimization options to compiler for facilitating inline expansions.
- Storing three FP21 data to two 32bits integer is new optimization.
  - In the previous study of FP21, authors are store three FP21 data to 64bits integer.
  - Number of computations per one SIMD instruction is capped by the widest data format.

One 64bits integer : 8 data
Two 32bits integer : 16 data
per one 512bits SIMD

Storage Format

Evaluating storage format



CRS

- Basic storage format for sparse matrix

ELL

- Considering a vector and SIMD operation
- Equalize number of non-zero elements in each row
- 0-padding to columns lacking non-zero elements

Sell-C-σ

- Proposed considering the SIMD operation
- Shape determined from parameter Chunk size C and scope σ
- Less 0-Paddings than ELL
- Improved cache hit ratio

*Kreutzer, M., et al "A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units", SIAM Journal on Scientific Computing,

Numerical environments

Env 1：Oakforest-PACS (OFP)

- Xeon Phi
  - 64 cores,128threads, MCDRAM
- Intel compiler (v19.1.1.304)
  - Options : -O3 -xMIC-AVX512 -qopenmp -align array64byte -ipo
  - Numerical environments: KMP_HW_SUBSET=64c@2,2t

Env 2：Oakbridge-CX (OBCX)

- Xeon Gold Platinum 8280 × 2
  - 56cores, 56threads, DDR4
- Intel compiler (v19.1.1.304)
  - Options：-O3 -xHost -qopenmp -align array64byte -ipo

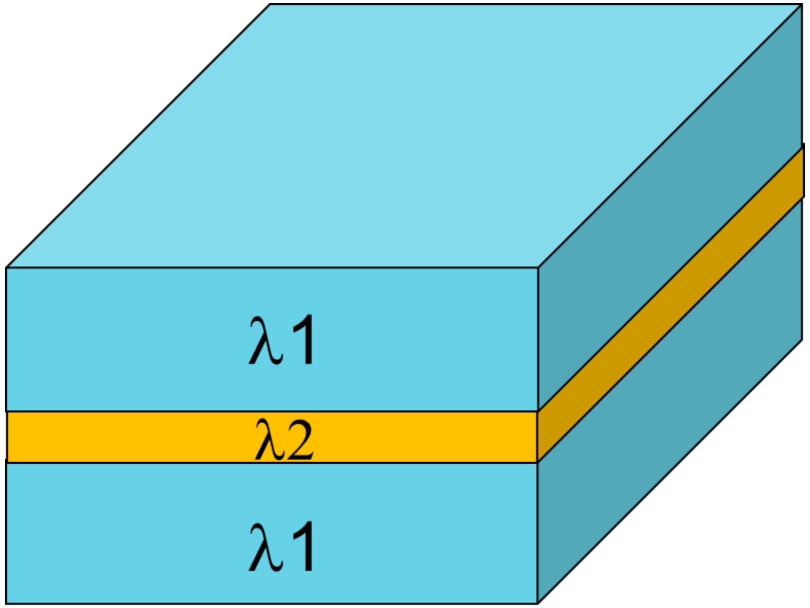
Env3：Wisteria/BDEC-01 Odyssey (WO)

- A64FX
  - 48cores, 48 threads, HBM2
- Fujitsu compiler (4.5.0 tcscd-1.2.31)
  - Options : -O3 -Kfast,openmp,zfill,A64FX,ARMV8_A
  - Numerical environments : FLIB_FASTOMP=TRUE, FLIB_HPCFUNC=TRUE, XOS_MMM_L_PAGING_POLICY=demand:demand:demand

Conditions of application (P3D)

P3D application

- DoF : 256^3 = 16,777,216
- Thermal diffusivity : $\lambda 1 = 1,\ 1 \le \lambda 2 \le 10^{10}$



ICCG solver

- Parallelized IC preconditioner with multi-coloring approach
  - Cyclic Multi-coloring + Reverse Cuthill-Mckee（CM-RCM）
  - Number of colors for CM-RCM : 10 colors
  - Convergence condition is $\frac{\|r^k\|_2}{\|r^0\|_2} \le 10^{-8}$
  - Storage formats of the matrices are CRS, ELL and Sell-C-σ
- Combination of the data formats of the matrix and vector
  - ✓ FP64-FP64
  - ✓ FP42-FP64
  - ✓ FP32-FP64
  - ✓ FP64-FP32
  - ✓ FP32-FP32
  - ✓ FP21-FP32
  - ✓ FP16-FP32

In descending order of the amount of memory transfer

Blue：Only evaluate on OFP，OBCX
Green：Only evaluate on WO

＊FP16 vector is not included because it dose not converged.

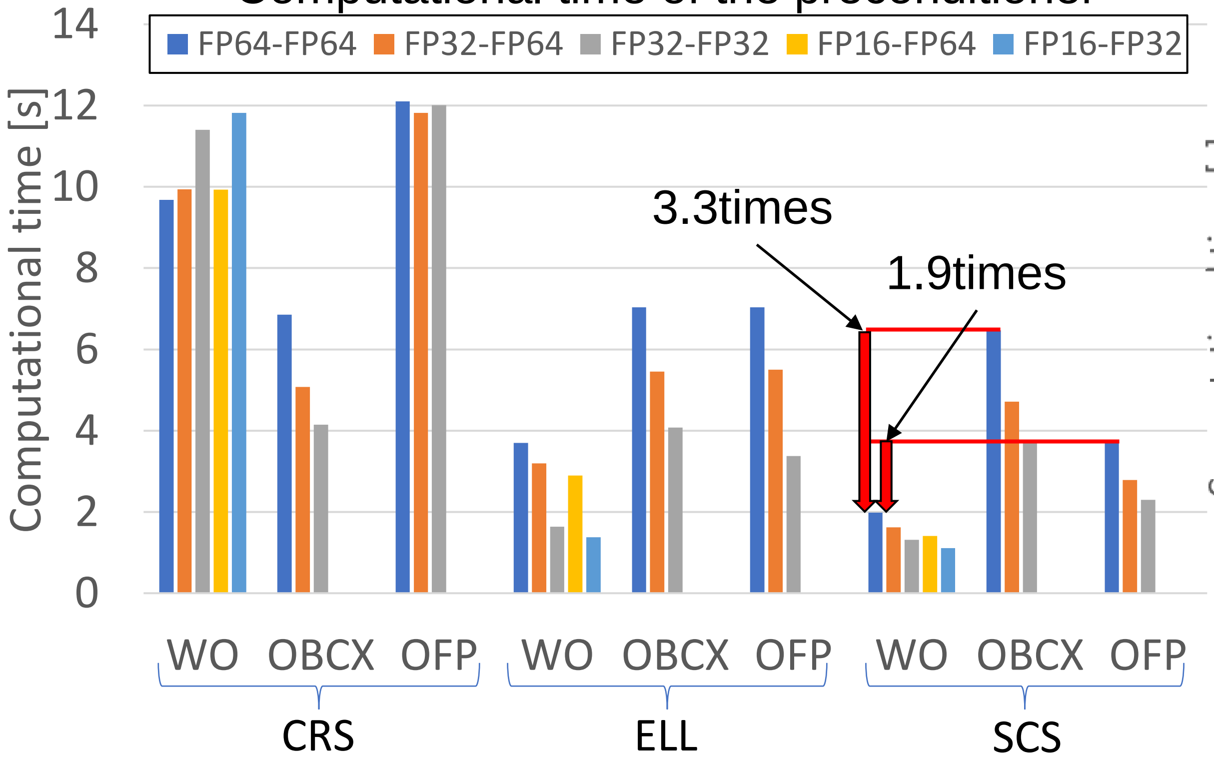
Denoted as data format of “matrix”-”vector”

Comparison among the storage formats

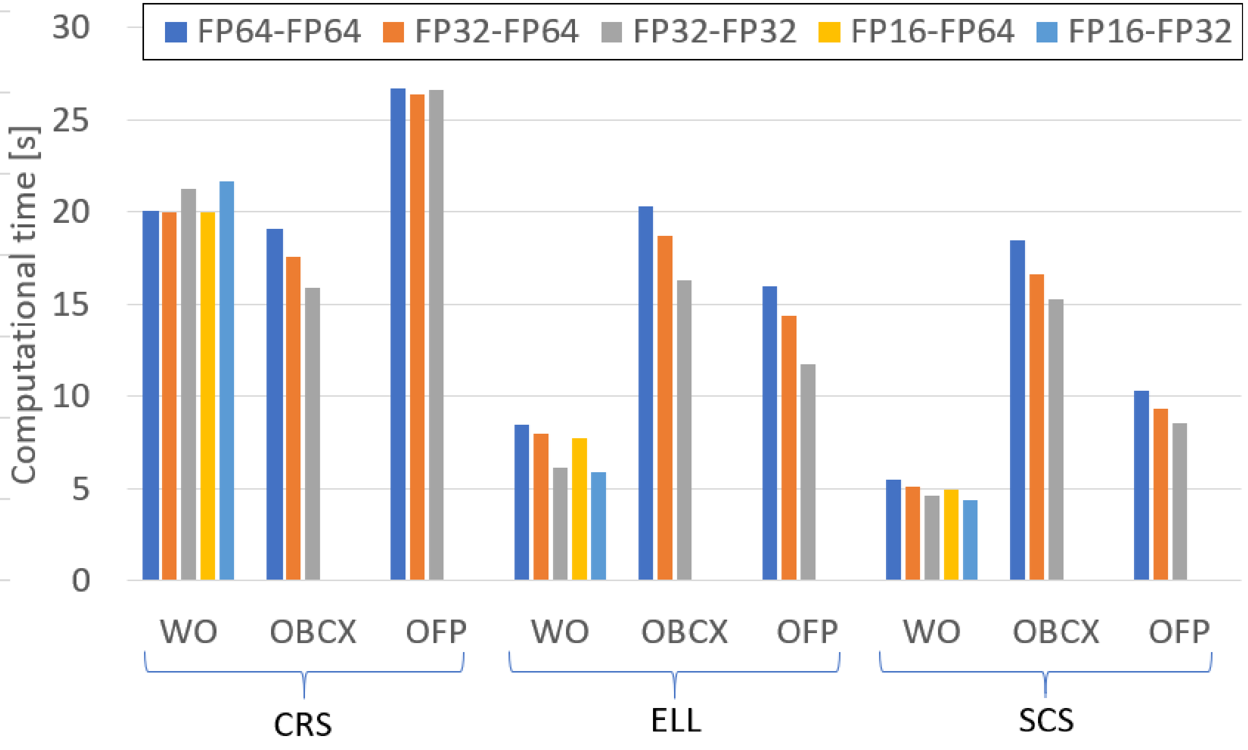
Sell-C-σ（FP64-FP64）shows performance close peak of memory bandwidth

- Computational time of Sell-C-σ（FP64-FP64）on WO is 3.3 and 1.9 times faster than OBCX and OFP, respectively.
  - Sell-C-σ（FP64-FP64）on WO shows 812GB/s (measured by Fujitsu profiler).
  - Stream triad : OBCX=280GB/s、OFP=490GB/s、WO=840GB/s (WO/OBCX=3.0, WO/OFP=1.7)

Computational time of the preconditioner



Whole computational time of ICCG

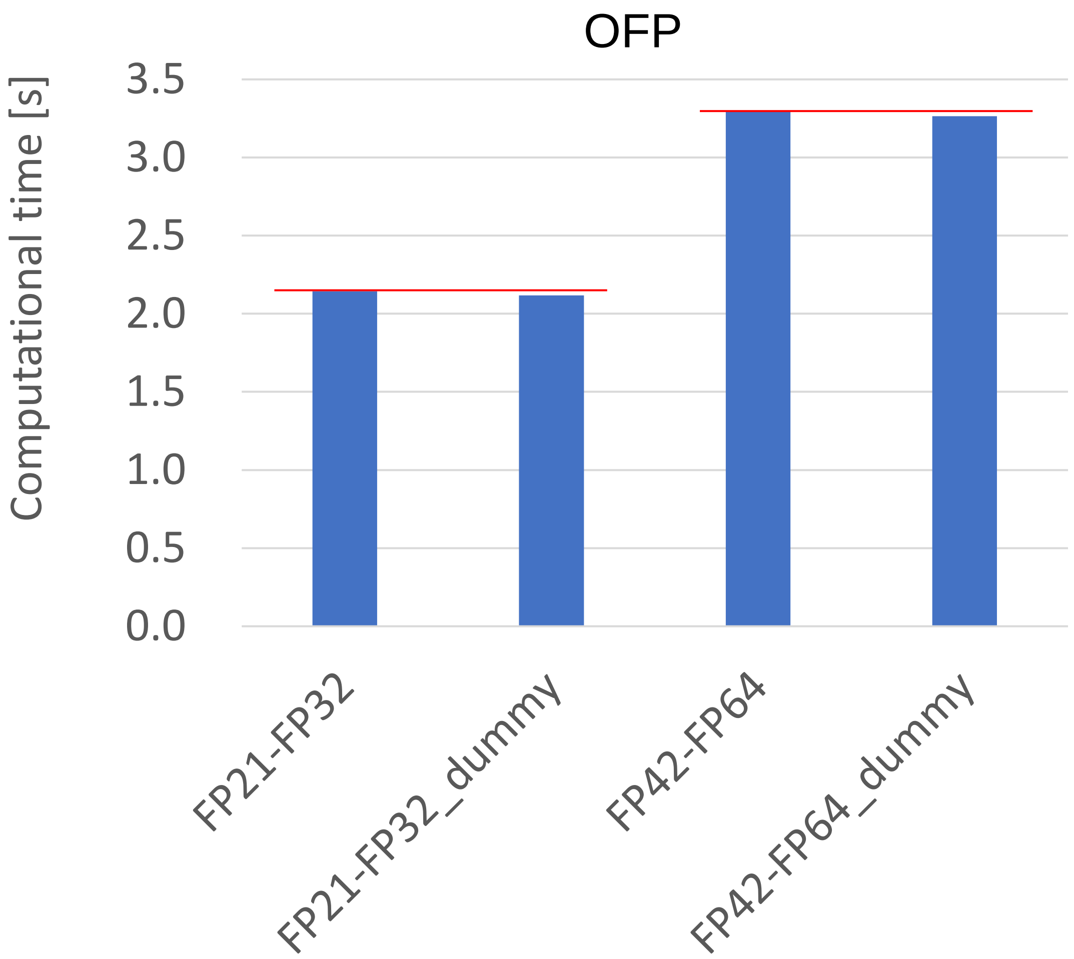


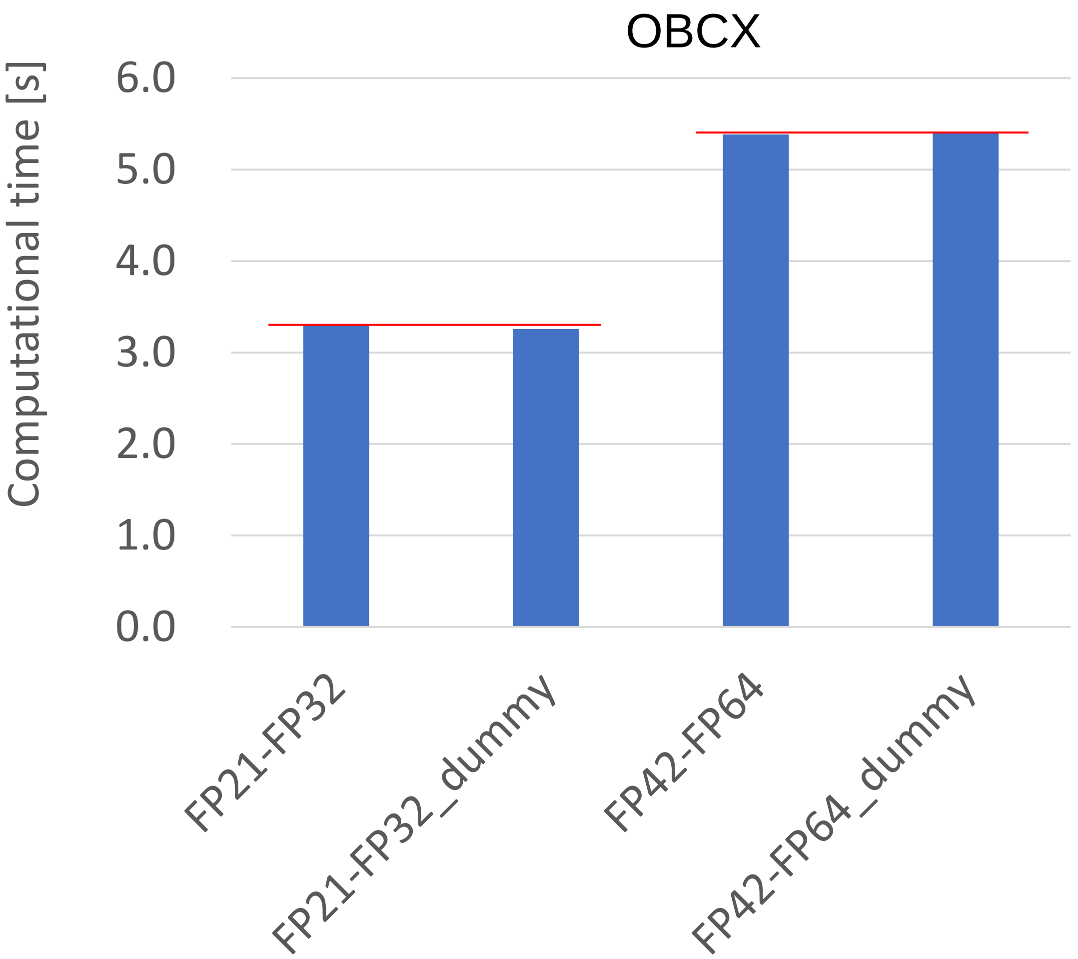
SCS = Sell-C-σ

Overhead of type casting of adaptive precisions

The overhead of typecasting is enough small.（Up to 1.5%)

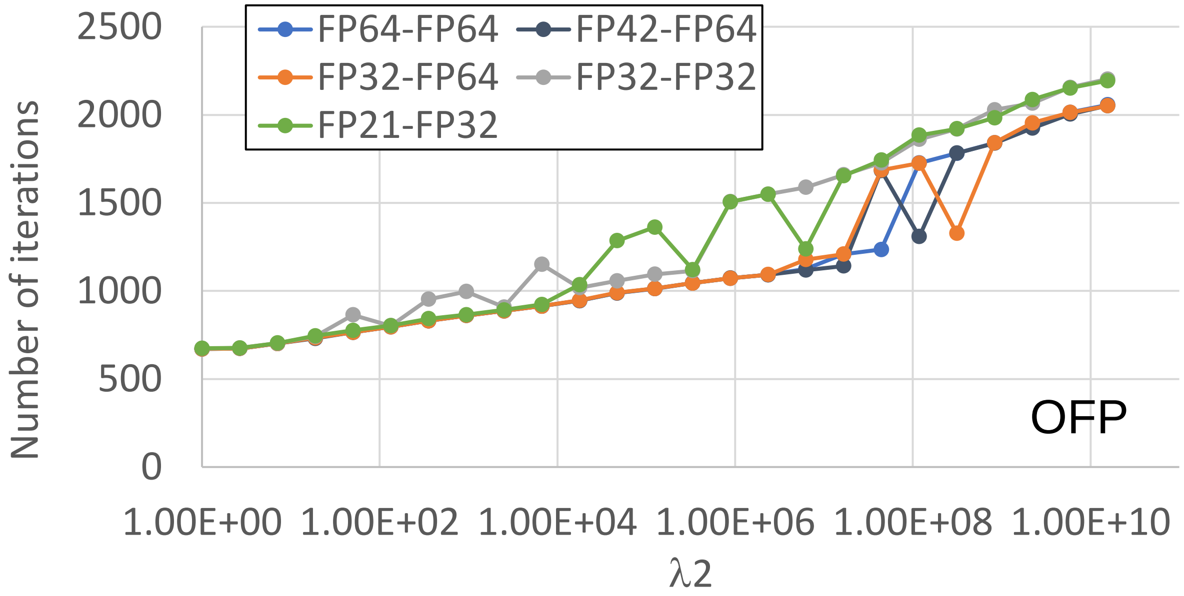
For measuring the overhead of typecasting, we prepared a dummy code that changed the FP21 or FP42 loading function to normal loading with the same amount of reference data.



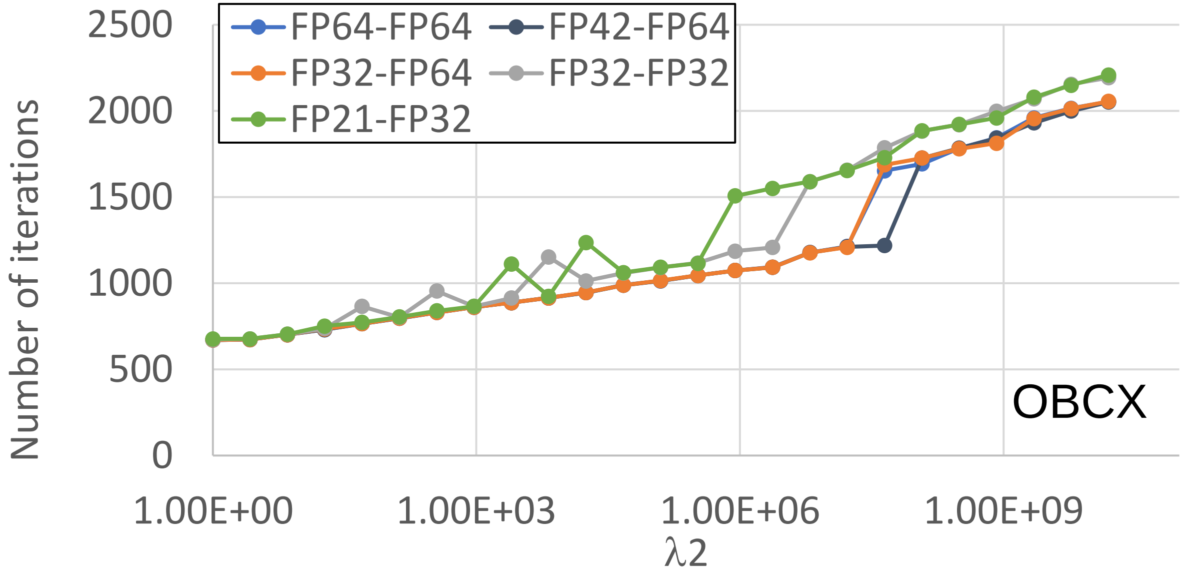


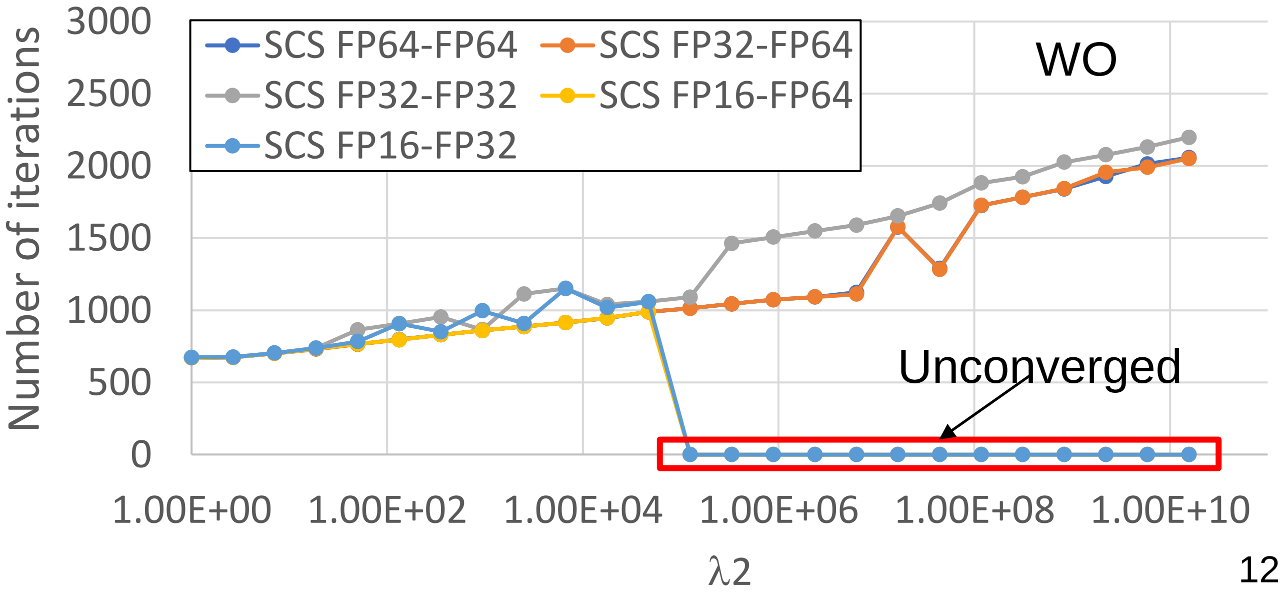
The difference between data format on convergence ratio

Different combination of data formats shows different convergence ratio.



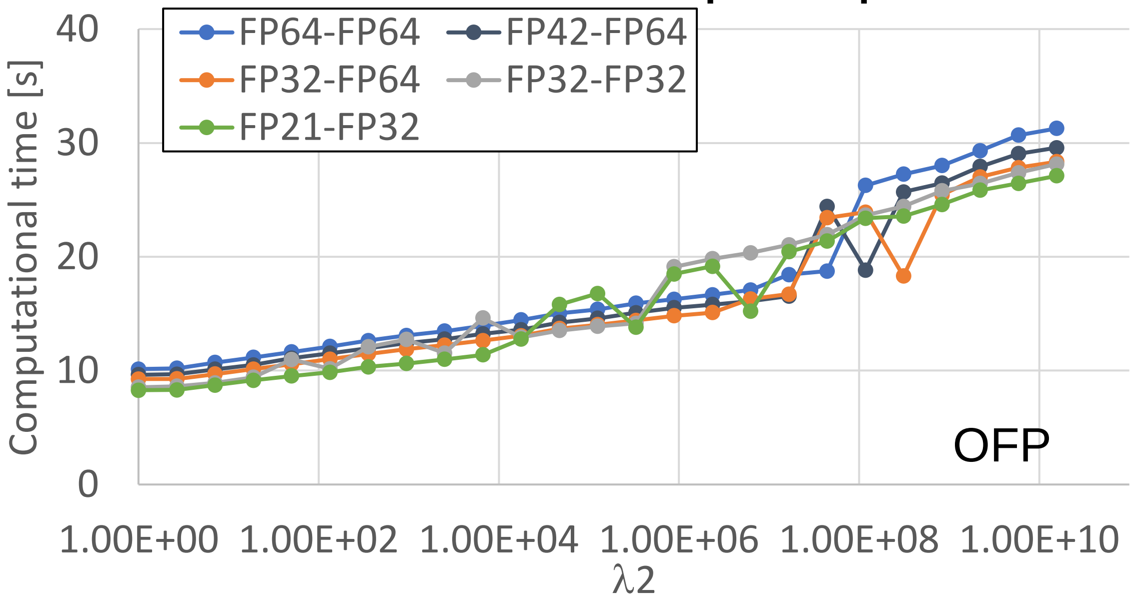
- There is no impact of lower data-precision with good conditions.
- FP32-FP16 is not converged with condition $\frac{\lambda_2}{\lambda_1} > 10^5 \rightarrow$ Beyond expression ability of FP16
- Convergence ratio get worse on ill-condition by changing vectors FP64→FP32



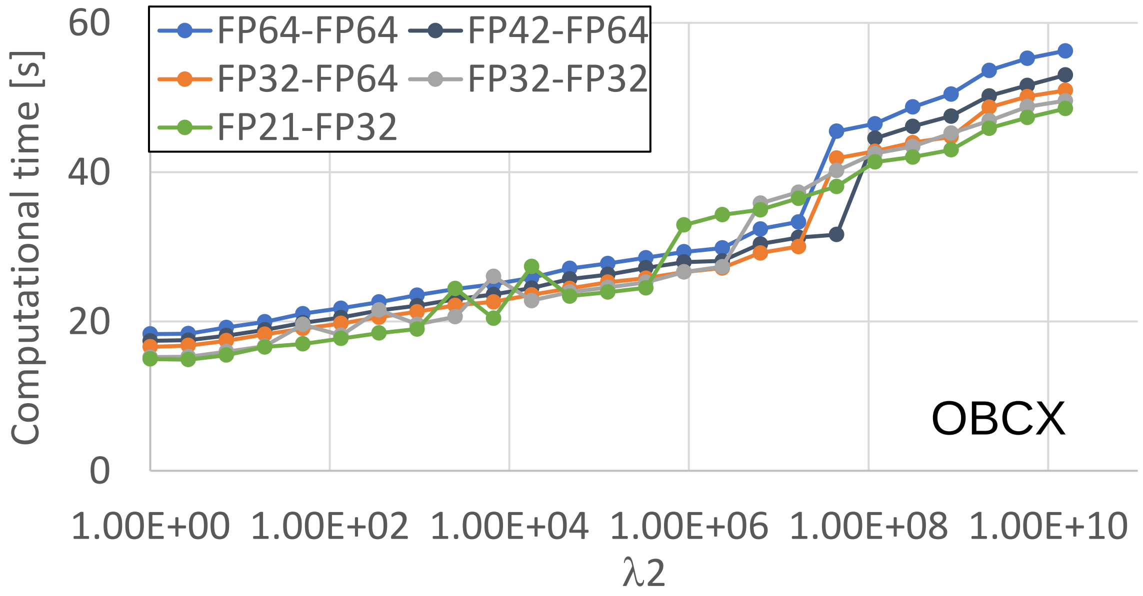


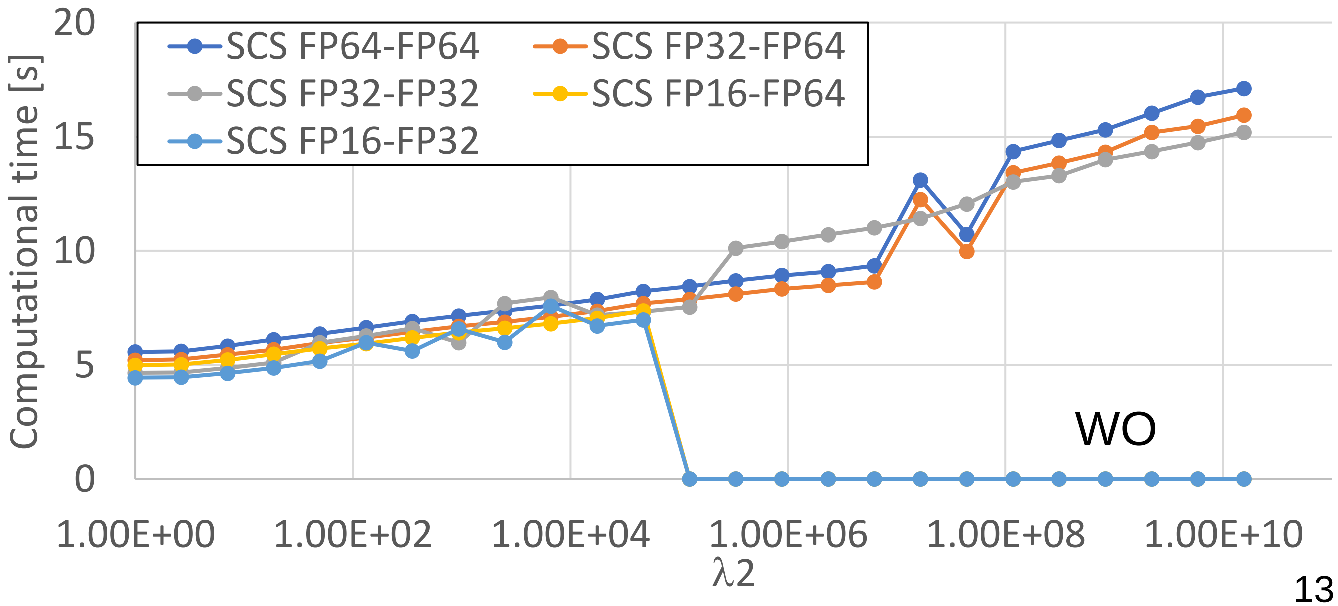
Performance improvement by low/adaptive precisions

Low/Adaptive precision shows reduce computational time.



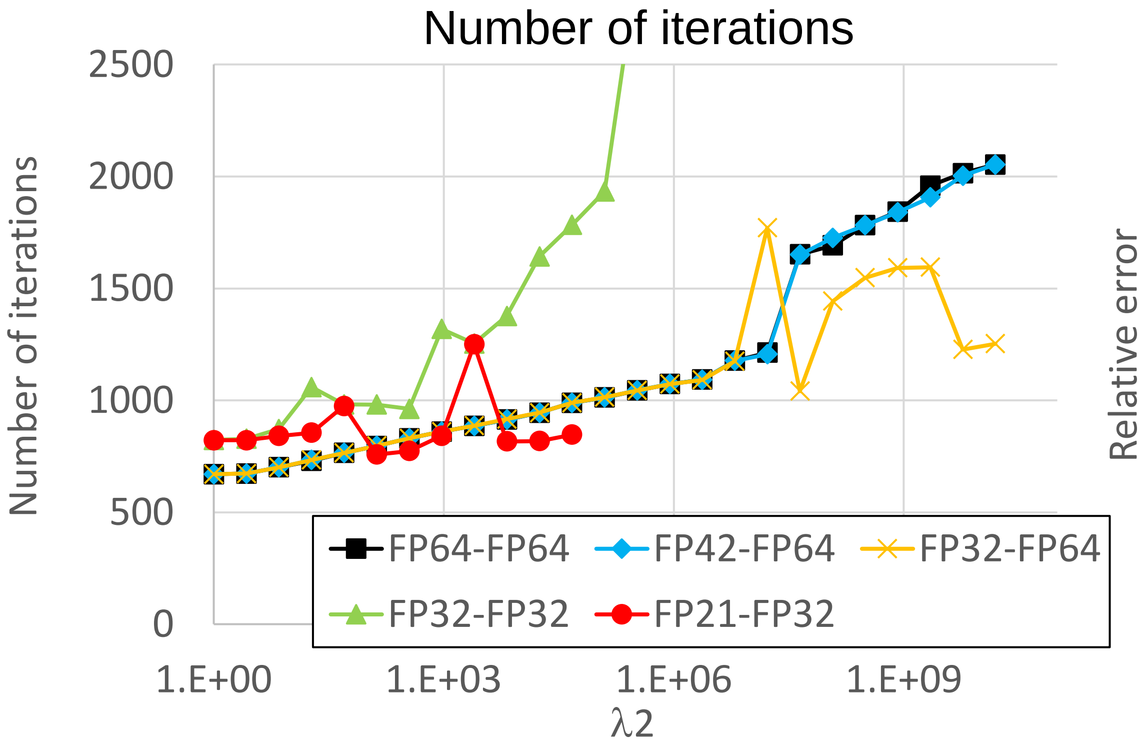
- FP16-FP32 was the fastest within the good condition.
  - 17.3% compared with FP64-FP64
- FP21-FP32 was the fastest within the good condition. on OFP and OBCX.
  - 18.4%(OFP), 18.6%(OBCX)
- FP32-FP64 was the fastest in intermediate conditions.
- FP21-FP32 was faster in worse condition, again.
  - 12.6%(OFP), 13.7%(OBCX)

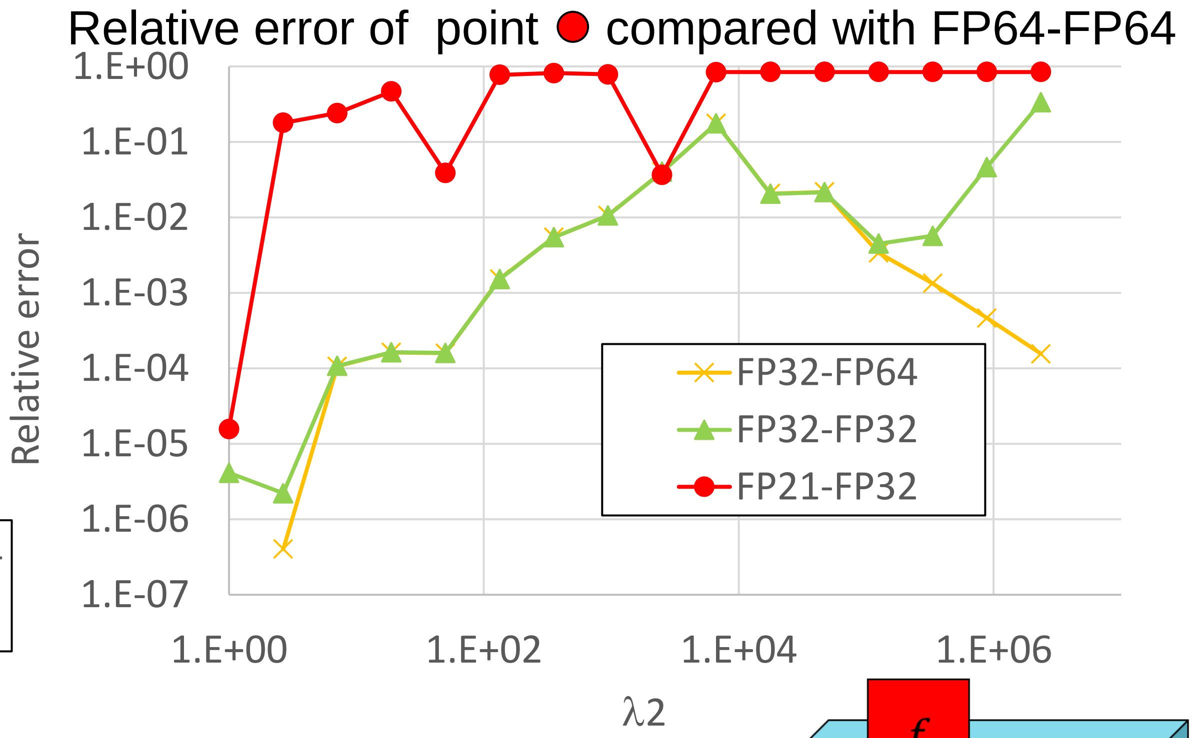




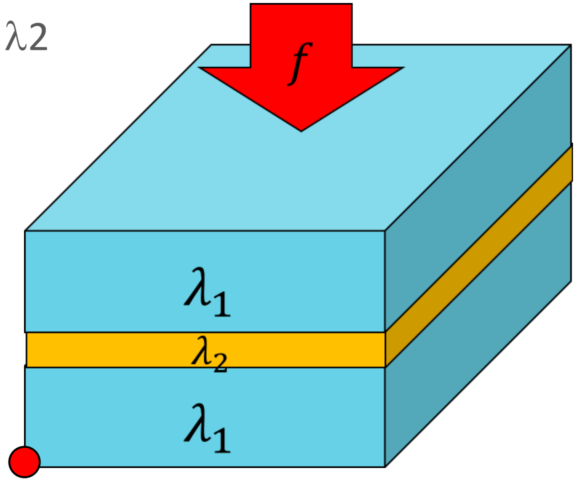
Applying low/adaptive precisions to whole ICCG 1/2

Relative error with FP21 and FP32 are large





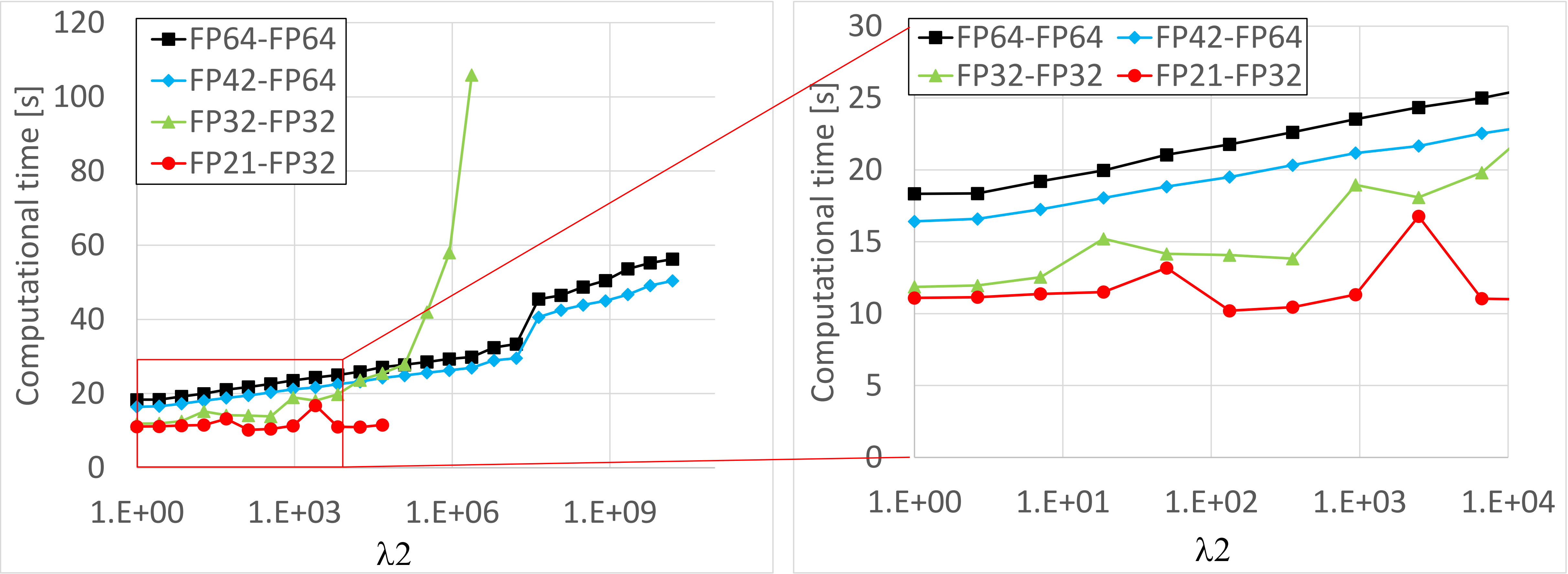
- The relative error of FP21-FP32 is more than 10^{-1} with $\lambda 2 > 2.66$
- The relative error of FP32-FP32 has reached to 10^{-1} with $\lambda 2 > 200$
- No deterioration of FP42-FP64 accuracy was observed.



Result of applying low or adaptive precision to all arrays

Environment : OBCX
DoF : 256^3 = 16,777,216
Storage Format : Sell-C-σ
Coloring : CM-RCM(10)

Applying low or adaptive precision to all matrices and vectors
FP42-FP64 is 10.5% faster than FP64-FP64 ($\lambda 2 = 1.0$).
FP32-FP32 is 35.3% faster than FP64-FP64 ($\lambda 2 = 1.0$).
FP21-FP32 is 39.5% faster than FP64-FP64 ($\lambda 2 = 1.0$).



Conclusion

- Evaluate the usefulness of low precision such as FP32 and FP16 and adaptive precision such as FP42 and FP21 in real applications where the use of FP64 is typical.
  - We choose the P3D for the evaluations as the real application.
  - ICCG solver is included in the P3D and it is a typical application using FP64.
- We optimize the load and store routine of FP21 on CPUs for general purpose.
  - We change a storing data type of FP21 from one 64bits integer to two 32bits integers.
- In the numerical evaluations, we apply low/adaptive precisions to the IC preconditioner part or whole ICCG method.
- The use of low/adaptive precision improves performance of ICCG method.
  - The effectiveness of Low/adaptive precision is high.
  - When we apply low/adaptive precision to the whole ICCG method, we have to consider the error of the result.
    - If the accuracy of the result is acceptable, low/adaptive precision shows good performance improvement.
  - The fastest combination of the matrix and vector is changed depending on the condition of the coefficient matrix.

Future work

- Considering an auto-tuning approach to dynamically select the best data format.
- Evaluation of more mixed precision : FP21-FP32 IC, FP42-FP64 CG part
- Adding verification