Low/Adaptive Precision Computation in Preconditioned Iterative Solvers for Ill-Conditioned Problems

Masatoshi Kawai (ITC U-Tokyo)
International Worskhop on the Integration of (S+D+L) :
Toward Society 5.0 h3-Open-BDEC
2021/11/30
Online

Outline

1. Objective
2. Low/Adaptive precisions
3. P3D application (ICCG method)
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 the low/adaptive precisions

⇩

The use of the lower precision reduces execution time

Because of improving an efficiency of a SIMDization and reducing amount of memory transfer.

As same as practical simulations,

- The use of lower precision reduces the execution time.
- FP21 (arbitrary precision) is proposed and 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.

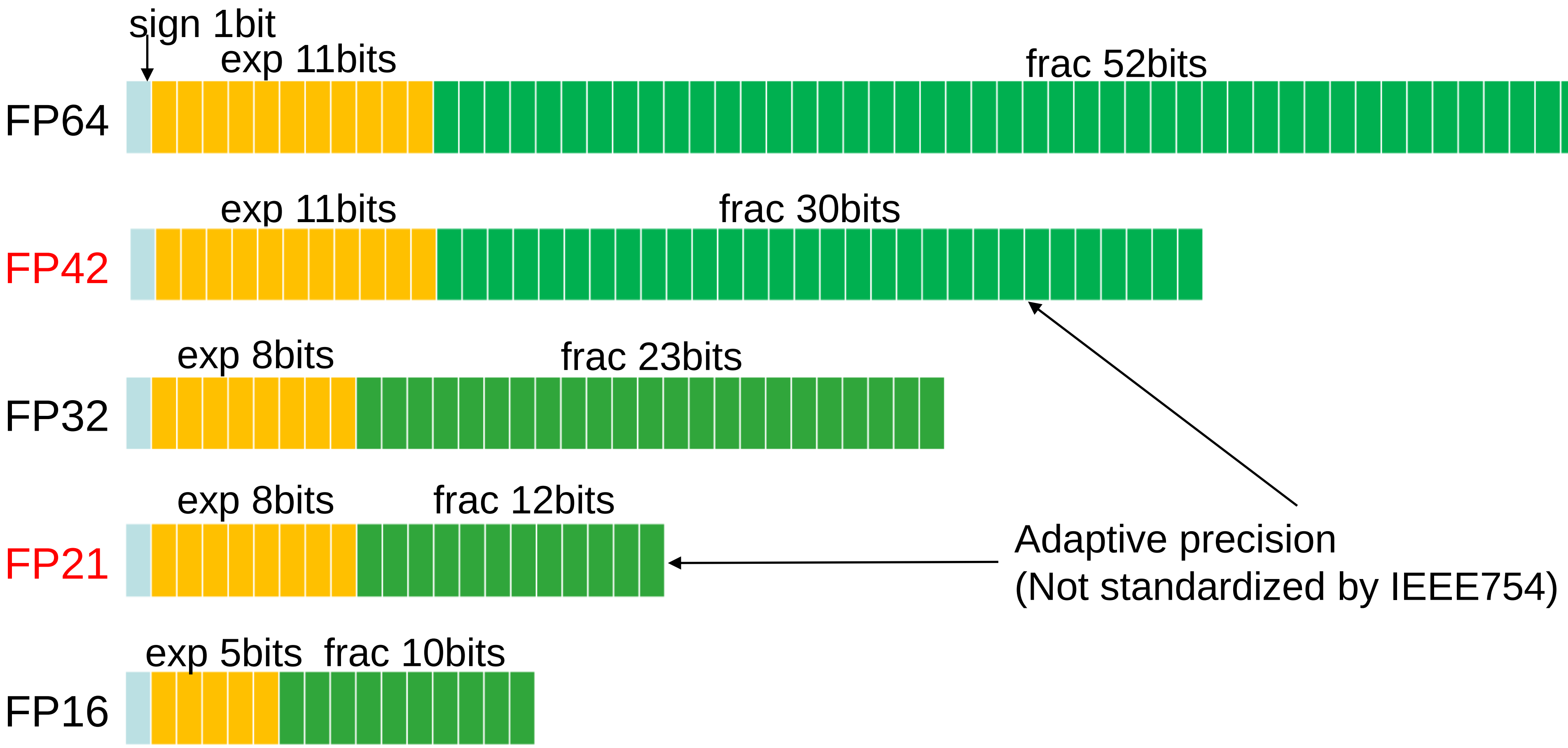
- ICCG is one of the most famous iterative method which require high accuracy of computations.
- The performance of the ICCG method is determined by memory bandwidth.

*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

Outline

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 subroutine/function 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

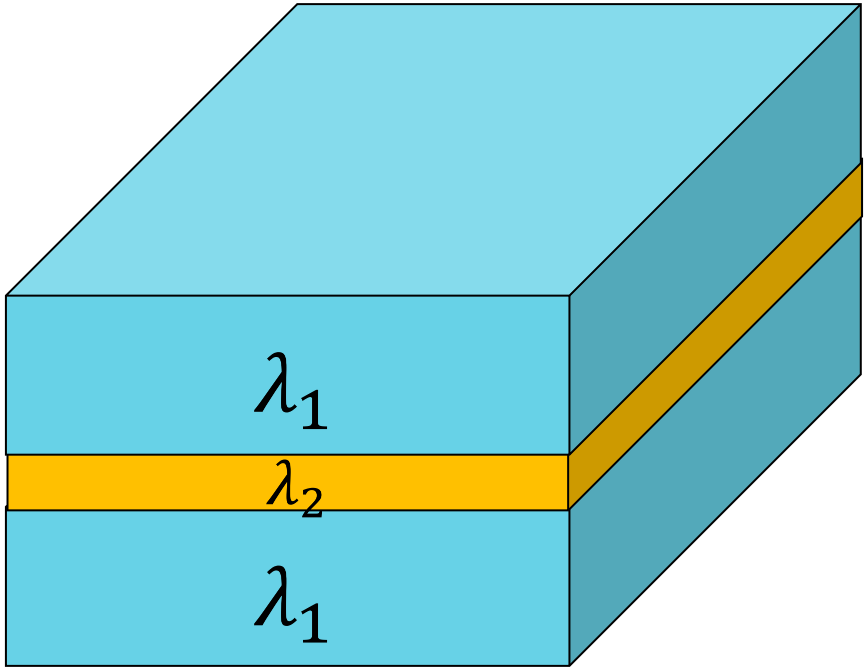
Outline

1. Objective
2. Low/Adaptive precisions
3. P3D application (ICCG method)
4. Numerical evaluations
5. Conclusion

P3D : Steady State 3D Heat Conduction by FVM

We use P3D application for numerical evaluations

- Simulation of 3D heat conduction
  - $\nabla \cdot (\lambda \nabla \phi) + f = 0$
  - Discretized by FVM
  - Seven-point stencil
- Boundary conditions
  - $\phi(X_{min})$、$\phi(X_{max})$、$\phi(Y_{min})$、$\phi(Y_{max})$、$\phi(Z_{min}) = 0$
  - $\phi(Z_{max}) = f$
- Factor λ : thermal diffusivity
  - A distribution of thermal diffusivity is showing right figure
    - ✓ $\lambda 1 = 1$、$1 \leq \lambda 2 \leq 10^{10}$
- ICCG solver
  - IC preconditioner is parallelized by multi-coloring method with CM-RCM algorithm
    CM-RCM : Cyclic-multicoloring + Reverse Cuthill Mckee



Coefficient matrix of P3D

The thermal diffusivity λ in the target problem has strong impact on a condition number.

$$a_{i,j} = \begin{cases} -\dfrac{dy \cdot dz}{\frac{dx}{2}\left(\frac{1}{\lambda_{x,y,z}} + \frac{1}{\lambda_{x-1,y,z}}\right)}、j = i - 1 \\ -\dfrac{dy \cdot dz}{\frac{dx}{2}\left(\frac{1}{\lambda_{x,y,z}} + \frac{1}{\lambda_{x+1,y,z}}\right)}、j = i + 1 \\ -\dfrac{dx \cdot dz}{\frac{dy}{2}\left(\frac{1}{\lambda_{x,y,z}} + \frac{1}{\lambda_{x,y-1,z}}\right)}、j = i - nx \\ -\dfrac{dx \cdot dz}{\frac{dy}{2}\left(\frac{1}{\lambda_{x,y,z}} + \frac{1}{\lambda_{x,y+1,z}}\right)}、j = i + nx \\ -\dfrac{dx \cdot dy}{\frac{dz}{2}\left(\frac{1}{\lambda_{x,y,z}} + \frac{1}{\lambda_{x,y,z-1}}\right)}、j = i - nx \times ny \\ -\dfrac{dx \cdot dy}{\frac{dz}{2}\left(\frac{1}{\lambda_{x,y,z}} + \frac{1}{\lambda_{x,y,z+1}}\right)}、j = i + nx \times ny \\ \sum_{k=1}^{N} -a_{i,k}、j = i \\ 0、\text{others} \end{cases}$$

If the factor λ in the target problem has large difference, diagonal and off-diagonal elements also have large difference.

→ We can control the condition number.

In this study, we change the factor λ_2 in numerical experiments for evaluating the difference among the data formats. (λ_1 is a constant)

ICCG method

We apply low/adaptive precision to the IC preconditioner

If we change the data format of the….

- coefficient matrix → the problem to be solved may change.
- vectors excluding $\acute{\boldsymbol{r}}$、$\boldsymbol{q}$ → the convergence ratio is changed significantly because of low accuracy of inner-products
- matrix $\acute{\boldsymbol{U}}^{-1}\acute{\boldsymbol{D}}^{-1}\acute{\boldsymbol{L}}^{-1}$ and vectors $\acute{\boldsymbol{r}}$、$\boldsymbol{q}$ for the IC preconditioner → it is efficient because of high computational cost and lower sensitivity to the convergence ratio.

Algorithm of ICCG

do k = 1, until converge

$$\alpha = \frac{(\boldsymbol{r}^k, \boldsymbol{p}^k)}{(\boldsymbol{p}, ^k \boldsymbol{A}\boldsymbol{p}^k)}$$

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha \boldsymbol{p}^k$$

$\boldsymbol{q}^k$: vector shows searching direction

$$\boldsymbol{r}^{k+1} = \boldsymbol{r}^k - \alpha \boldsymbol{A}\boldsymbol{p}^k$$

$\boldsymbol{r}^k$: residal vector

$$\acute{\boldsymbol{r}} = \boldsymbol{r}^k$$

$$\boldsymbol{q} = \acute{\boldsymbol{U}}^{-1}\acute{\boldsymbol{D}}^{-1}\acute{\boldsymbol{L}}^{-1}\acute{\boldsymbol{r}}$$

$$\boldsymbol{p}^{k+1} = \boldsymbol{q} - \frac{(\boldsymbol{q}, \boldsymbol{r}^{k+1})}{\rho}\boldsymbol{p}^k, \rho = (\boldsymbol{q}, \boldsymbol{r}^{k+1})$$

enddo

Applying arbitrary precisions to IC preconditioner

Considering two implementation to apply arbitrary precision to IC preconditioner
→ Row-wise and column-wise

Evaluating both implementations and choose better one on each system.

```
real(8) ALpre(DoF, NoC)

do j = 1, NoC
   do i = 1, DoF
      q(i) = q(i) − ALpre(i, j) ∗ rd(idx_colum(i, j))
   enddo
enddo
```

Row-wise

```
fp21x3 ALpre(DoF/3*2, NoC)
real(4) ALpre1, ALpre2, ALpre3

do j = 1, NoC
   do i = 1, DoF, 3
      k = (i – 1) / 3 * 2 + 1
      call fp21x3_to_floatx3_f(ALpre(k:, j), ALpre1, ALpre2, ALpre3)
      q(i)     = q(i)     − ALpre1 ∗ rd(idx_colum(i,     j))
      q(i+1) = q(i+1) − ALpre2 ∗ rd(idx_colum(i+1, j))
      q(i+2) = q(i+2) − ALpre3 ∗ rd(idx_colum(i+2, j))
   enddo
enddo
```

Column-wise

```
fp21x3 ALpre(DoF, NoC/3*2)
real(4) ALpre1, ALpre2, ALpre3

do j = 1, NoC, 3
   k = (j – 1) / 3 * 2 + 1
   do i = 1, DoF
      call fp21x3_to_floatx3_f(ALpre(i, k:), ALpre1, ALpre2, ALpre3)
      q(i) = q(i) − ALpre1 ∗ rd(idx_colum(i, j  ))
      q(i) = q(i) − ALpre2 ∗ rd(idx_colum(i, j+1))
      q(i) = q(i) − ALpre3 ∗ rd(idx_colum(i, j+2))
   enddo
enddo
```

Outline

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

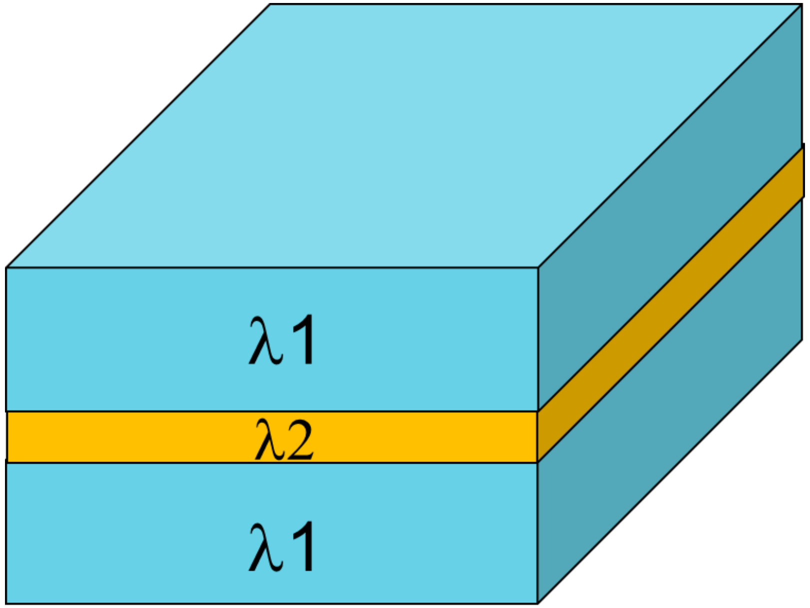
Env3：Wisteria/BDEC-01 Odyssey (WO)

- 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 : $\lambda 1 = 1,\ 1 \leq \lambda 2 \leq 10^{10}$



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 $\frac{\|r^k\|_2}{\|r^0\|_2} \leq 10^{-8}$
  - Storage format of the matrices is 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

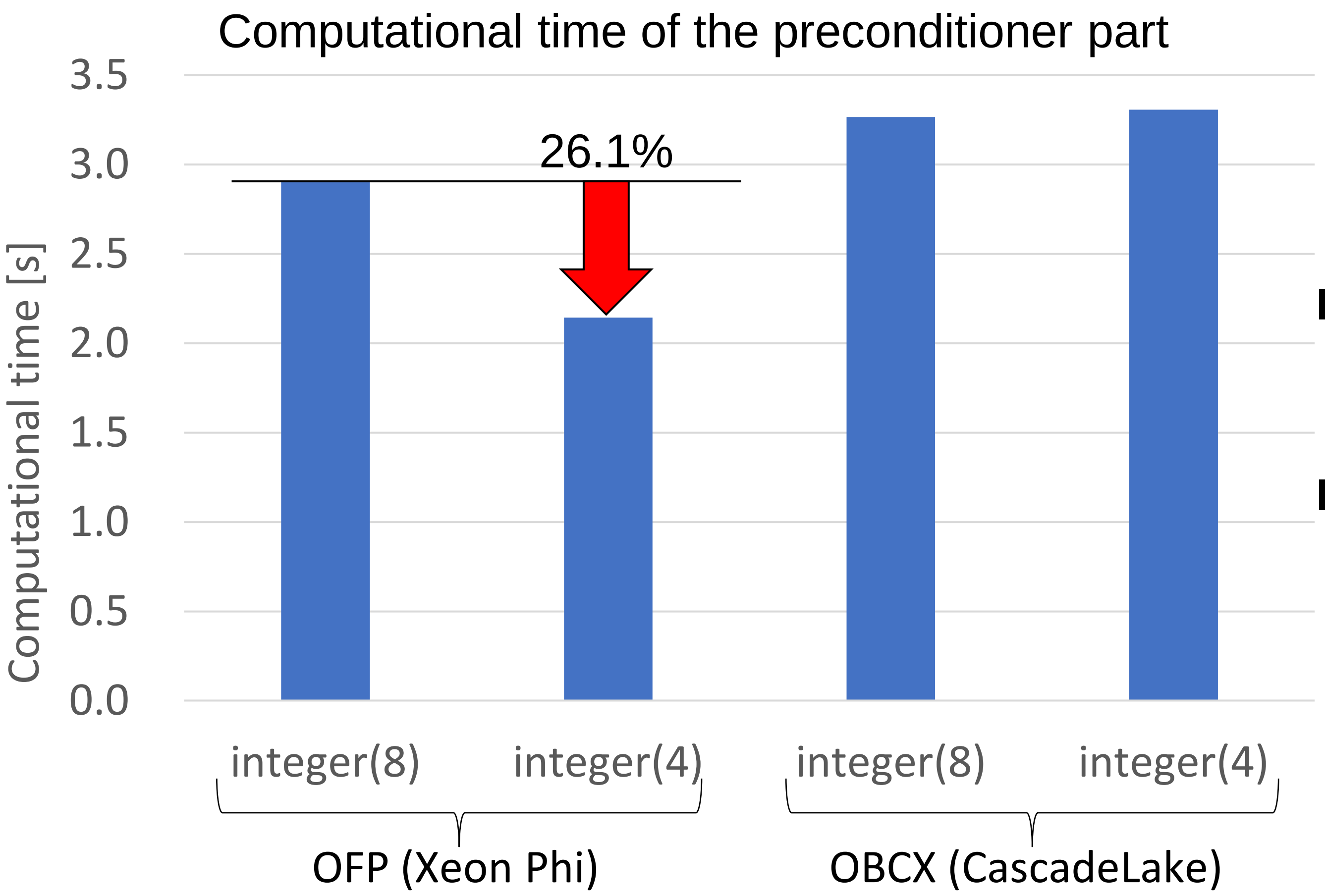
Denoted as data format of “matrix”vector”

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

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

Efficiency of two 32bits integers storing of FP21

Storing FP21 by two 32bits integer improves a performance by 26.1%.

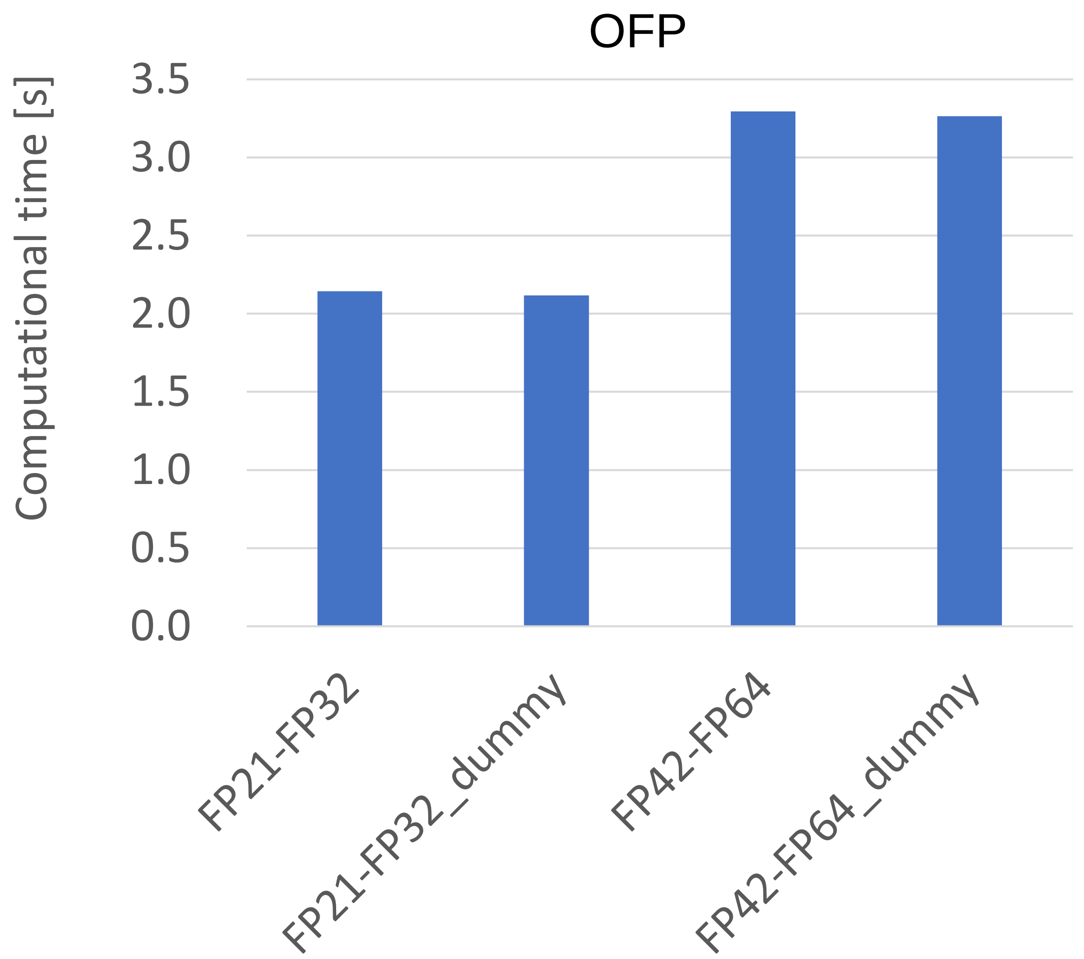


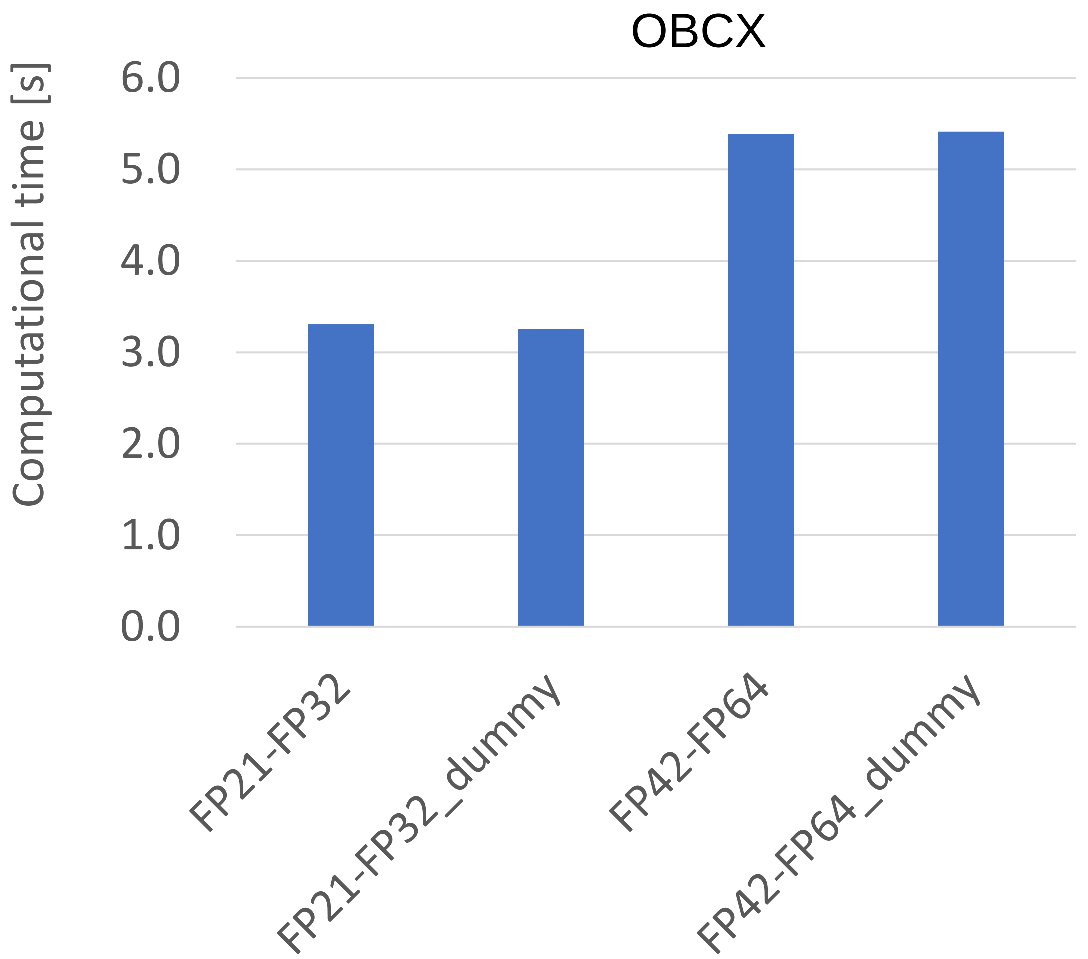
- Storing FP21 by two 32bits integers have shown 26.1% performance improvement on OFP.
- Less effective on OBCX

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.

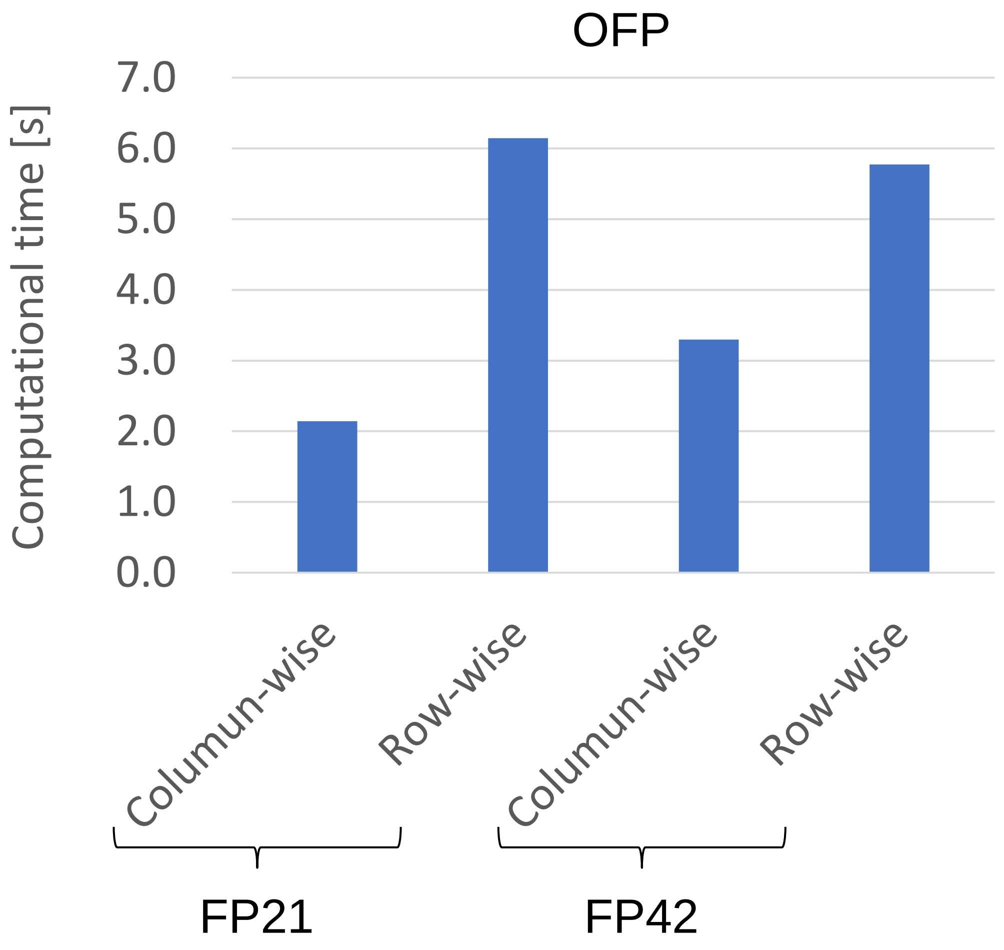


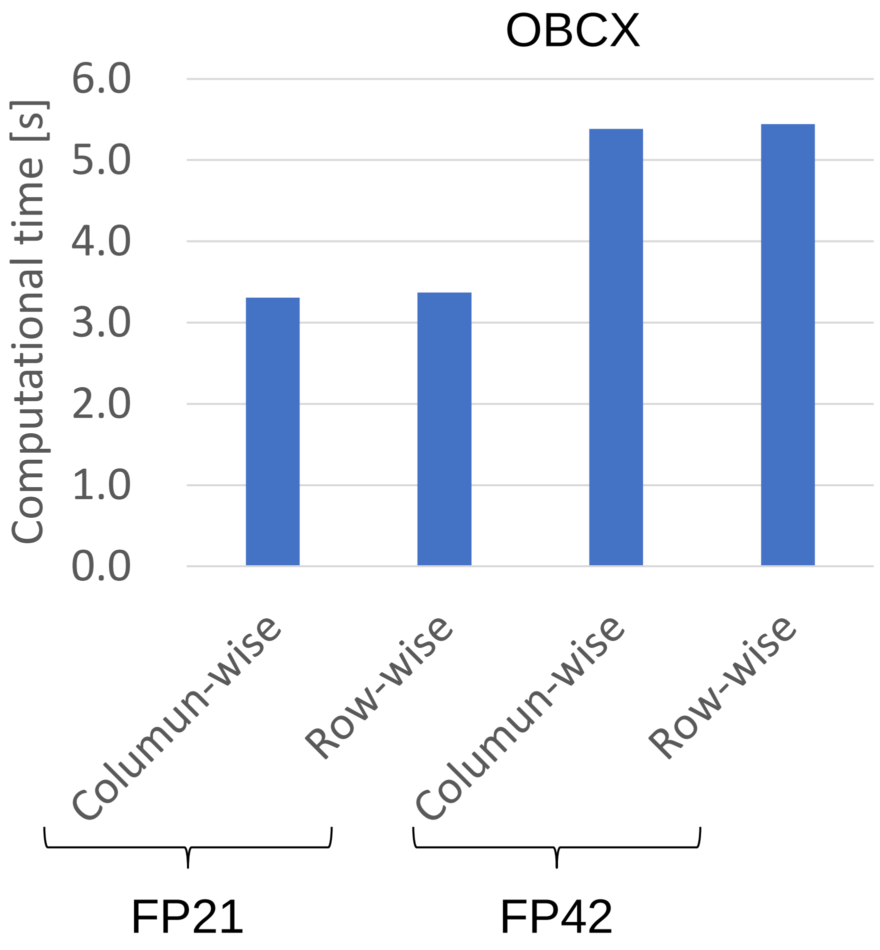


Comparison between Column-wise and Row-wise expansion

Column-wise expansion for implementing adaptive precision is faster than the row-wise.

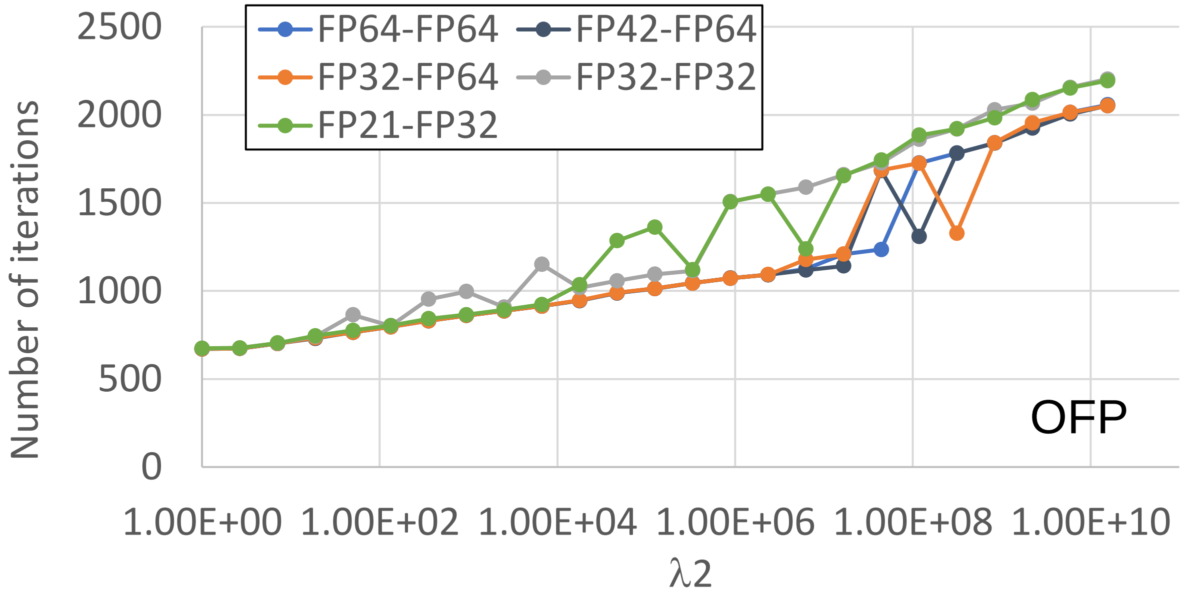
→ We use column-wise implementation for following evaluations.



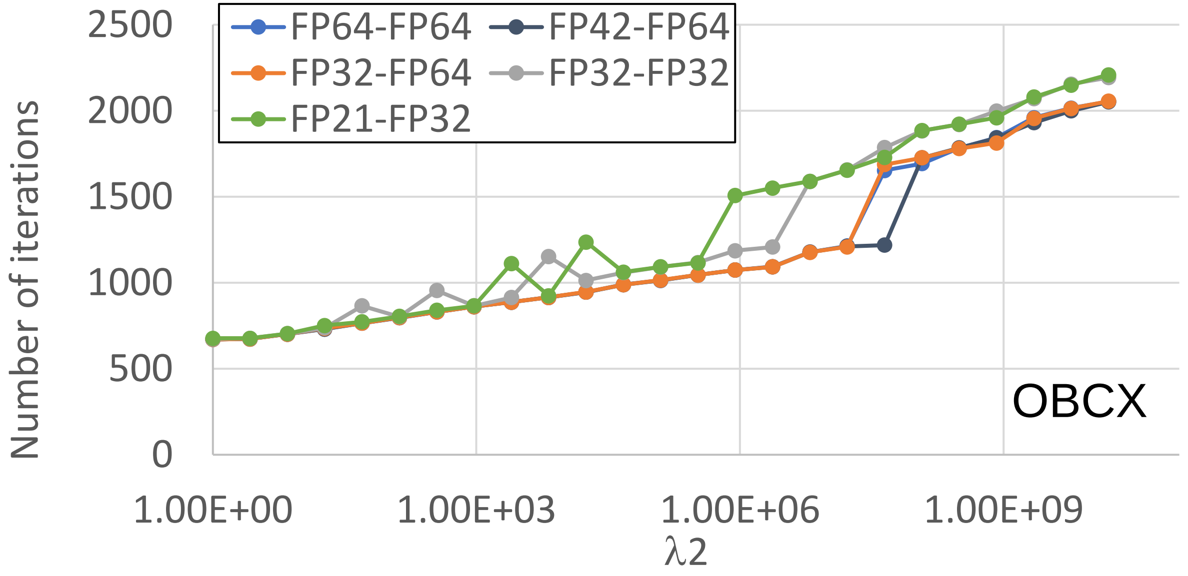


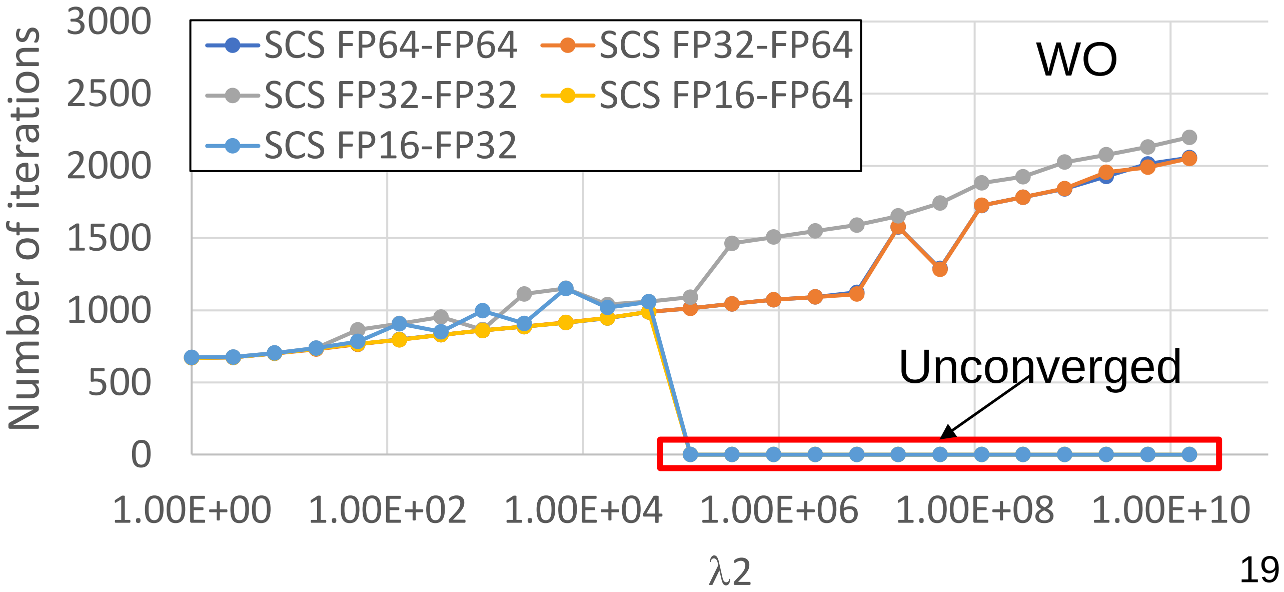
The difference between data format on convergence ratio

Different combination of data formats shows different convergence ratio.



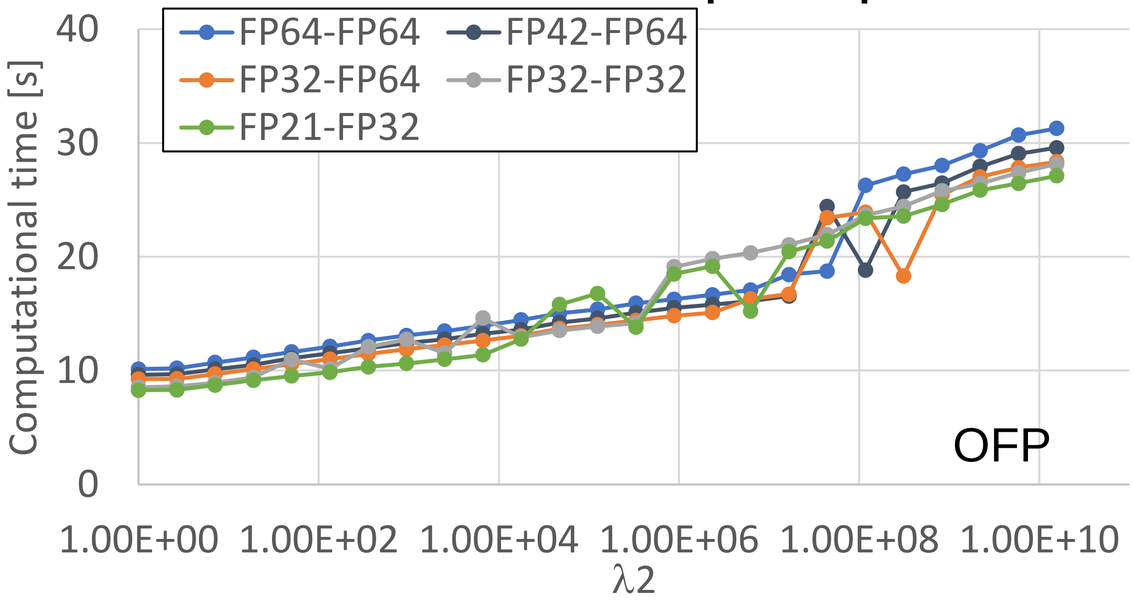
- There is no impact of lower data-precision under good conditions.
- FP32-FP16 is not converged with condition $\frac{\lambda_2}{\lambda_1} > 10^5 \rightarrow$ Beyond expression ability of FP16
- Convergence ratio get worse on ill-condition by changing vectors FP64→FP32



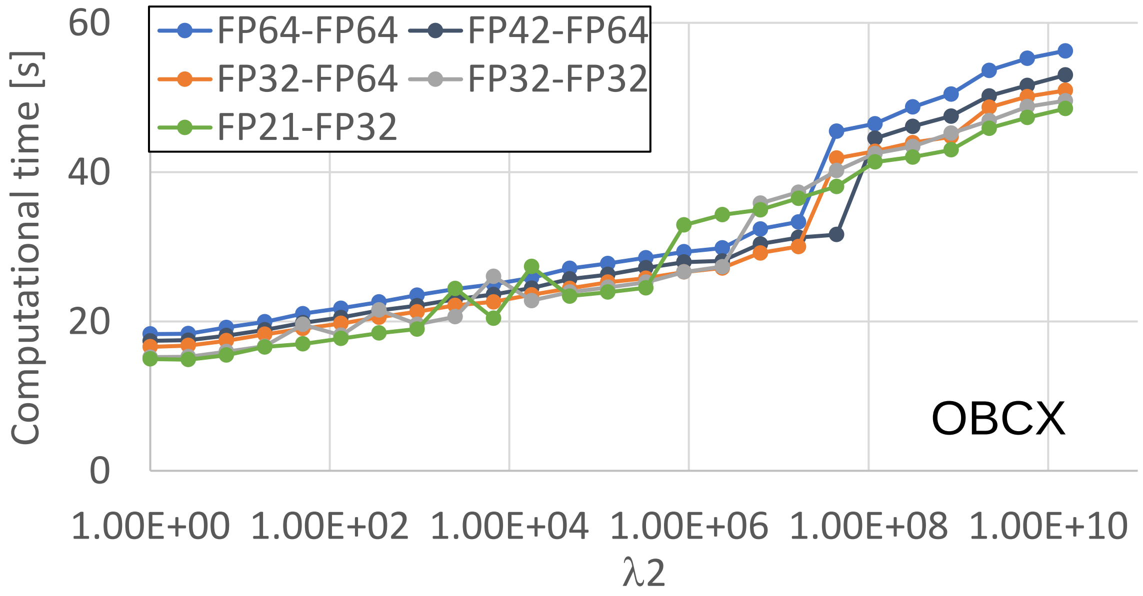


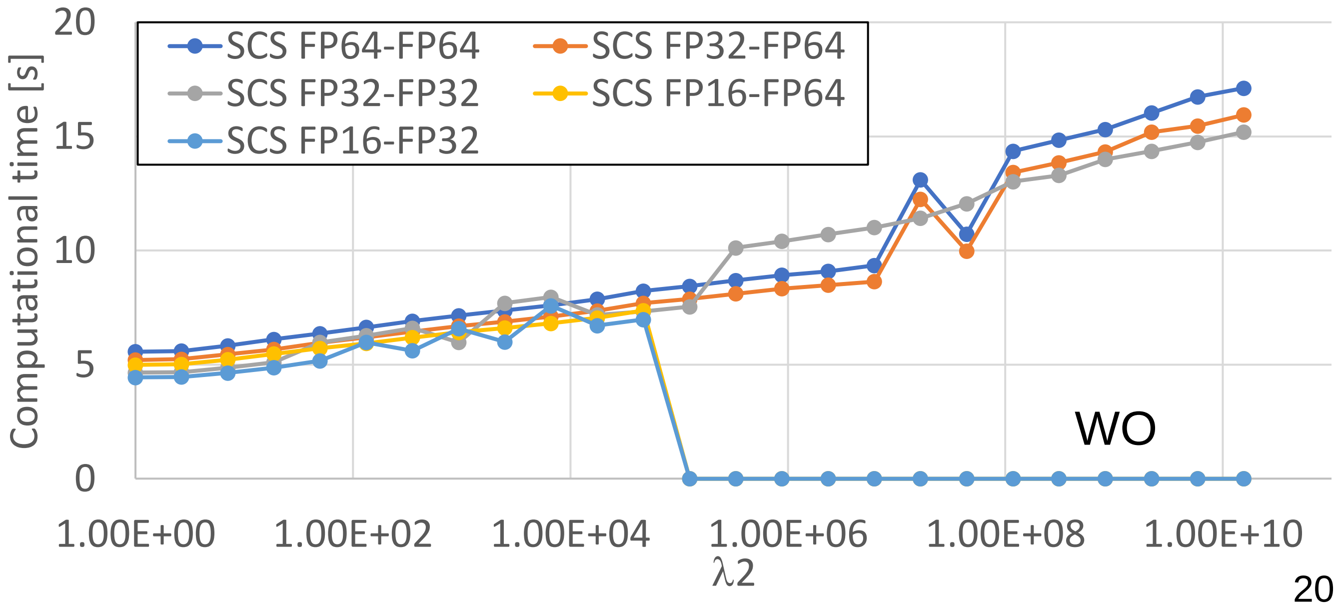
Performance improvement by low/adaptive precisions

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)





Outline

1. Objective
2. Low/Adaptive precisions
3. P3D application (ICCG method)
4. Numerical evaluations
5. Conclusion

Conclusion

- Evaluate the usefulness of low precision such as FP32 and FP16 and arbitrary 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 an IC preconditioner part.
  - The preconditioner part is implemented with Sell-C- storage format.
- The use of low/adaptive precision improve performance of ICCG method.
  - The effectiveness of Low/adaptive precision is high within the good conditions and expressible range of FP16
  - 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 precision.