PRC 2022
Final presentation

Haderbache Amir, Satoru Kawaharai
Professors: Sasa-sensei, Okuda-sensei
11/28/2022

Experimentation with FrontISTR

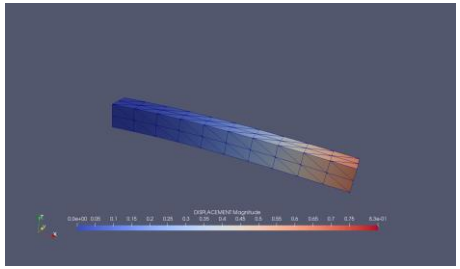
- Experimental Environment
- CAE Simulation model
- Simulation parameter + visualization results

Experimental Environment

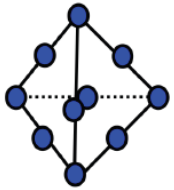
- Intel machine
  - **CPU:** Intel Xeon E5 (36C / 72T)
  - **Memory:** 128 GB DDR4 RAM
  - **Storage:** local Intel NVMe SSD 750 Series 1.2TB

- FrontISTR build
  - **Version**: 5.4 (latest)
  - **Compilers:** Intel compilers (mpiifort, mpiicc, mpiicpc) version 20.2.6.20220226
    - **options**: –O3 –xhost
  - **Build with cmake 3.16**
    - Intel OpenMP 5.1 , Intel MPI 3.1, Scalapack-2.1.0, Intel MKL 2022.1.0 (BLAS, LAPACK)
    - Metis 5.1.0, Trilinos 13.0.1

FrontISTR Simulation: 3D Beam model



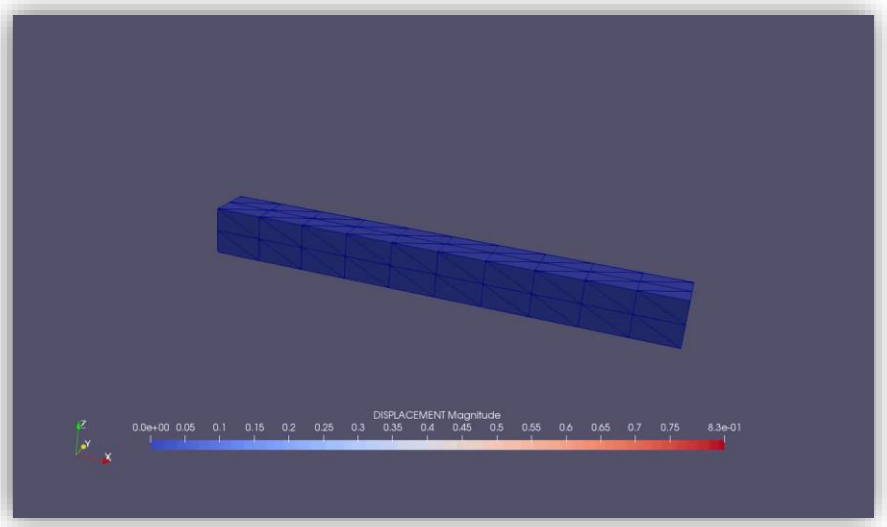
- 3D Beam model
  - Application: tutorial, material fatigue
  - Mesh: 524 nodes
  - Element: 240 elements
  - Element type: = 10-nodes tetrahedral quadratic (ID=342)
  - Material: Elastic Solid
    - Young modulus=4000
    - Poisson ratio=0.3
    - Density=1.0E-8
  - Boundary conditions:
    - left part is fixed
    - concentrated load is applied to other part with given amplitude



342

FrontISTR Simulation: 3D Beam model

- Simulation parameters
  - Analysis type: Dynamic Nonlinear
  - Numerical method: Newmark
  - Linear Solver: CG-DIAG
  - Linear tolerance: 1E-06
  - Nonlinear tolerance: 1E-03
  - Timesteps: 100,000
  - OMP Threads: 72, 36
  - Simulation computation time: 796.88 sec



Nonlinear dynamic analysis

- Governing Equation
- Algorithm
- Numerical Method

Governing Equation

- Dynamic equation of motion:

$$\boldsymbol{M\ddot{u}_n + r(u_n, \dot{u}_n) = f(t_n)}$$
$$\boldsymbol{u(t_0) = u_0, \quad \dot{u}(t_0) = \dot{u}_0}$$

with:

M: mass matrix

r: restoring force

f: external force vector

$u_n, \dot{u}_n, \ddot{u}_n$: displacement, velocity, acceleration at timestep t_n

$M\ddot{u}_n$: inertial force

$r(u_n, \dot{u}_n)$: damping + internal forces

Dynamic Nonlinear Implicit Algorithm

- Dynamic Nonlinear Implicit Algorithm:
  - Initialization
  - For each timestep {
    - For each nonlinear iteration {
      - Stiffness Matrix Generation
      - Linear Solver
      - Update Newton

      }
    - Write Results

  }

Newmark method

- Newmark method is used to compute next timestep acceleration (→ velocity and position)

1. $\ddot{u}_{k+1} \leftarrow 0$
2. $u_{k+1} = u_k + \dot{u}_k \, \Delta t + \ddot{u}_k \, (\frac{1}{2} - \beta)\Delta t^2 + \ddot{u}_{k+1}\beta\Delta t^2$
3. $\dot{u}_{k+1} = \dot{u}_k \; + \ddot{u}_k \, (1 - \gamma)\Delta t + \ddot{u}_{k+1}\gamma\Delta t$
4. $\boldsymbol{\varepsilon} \leftarrow \; f(t_{k+1}) \; - r(u_{k+1}, \dot{u}_{k+1}) - M\ddot{u}_{k+1}$ # $\boldsymbol{\varepsilon}$ is the nonlinear residual value
5. **while** $||\boldsymbol{\varepsilon}||$ >= nonlinear tolerance **do**
   i. $\Delta\ddot{u}_{k+1} \leftarrow (M + C\gamma\Delta t + K\beta\Delta t^2)^{-1} \boldsymbol{\varepsilon}$ # $x \leftarrow A^{-1} b$ ==> solving Ax=b requires a linear solver
   ii. $\ddot{u}_{k+1} \leftarrow \ddot{u}_{k+1} + \Delta\ddot{u}_{k+1}$
   iii. $\dot{u}_{k+1} \leftarrow \dot{u}_{k+1} + \Delta\ddot{u}_{k+1} \, \gamma\Delta t$
   iv. $u_{k+1} \leftarrow u_{k+1} + \Delta\ddot{u}_{k+1} \, \beta\Delta t^2$
   v. $\boldsymbol{\varepsilon} \leftarrow \; f(t_{k+1}) \; - r(u_{k+1}, \dot{u}_{k+1}) - M\ddot{u}_{k+1}$
6. **end while**

The numerical precision of simulation results $(u_n, \dot{u}_n, \ddot{u}_n)$ depends directly on the linear solver solution $(\Delta\ddot{u}_n)$.

FrontISTR and Numerical Precision

- Research scope
- Numerical Precision in FrontISTR
- Double precision FP number
- QPBLAS

Research Scope

- Focus on improving numerical precision of Linear Solver.
  ➔ will improve the numerical precision of CAE simulation.

- Target specific linear solver: the Conjugate Gradient method with Diagonal Scaling preconditioner (widely used in numerical simulation).

Numerical precision in FrontISTR

- In FrontISTR, linear solver are defined in the HECMW library.
- Linear solver data (matrix and vector) are defined in the hecmwST_matrix structure.

- The Fortran data type used for matrix/vector is real(kind=kreal) with kreal=8 → 8-bytes floating point number = 64-bits FP number = **double precision**

FrontISTR 5.4.0
Large-scale structural analysis program

Main Page | Modules | Data Types List | Files

integer(kind=**kint**)	**np**
integer(kind=**kint**)	**npl**
integer(kind=**kint**)	**npu**
integer(kind=**kint**)	**ndof**
integer(kind=**kint**)	**npcl**
integer(kind=**kint**)	**npcu**
real(kind=**kreal**), dimension(:), pointer	**d**
real(kind=**kreal**), dimension(:), pointer	**b**
real(kind=**kreal**), dimension(:), pointer	**x**
real(kind=**kreal**), dimension(:), pointer	**alu**
real(kind=**kreal**), dimension(:), pointer	**al**
real(kind=**kreal**), dimension(:), pointer	**au**
real(kind=**kreal**), dimension(:), pointer	**cal**
real(kind=**kreal**), dimension(:), pointer	**cau**

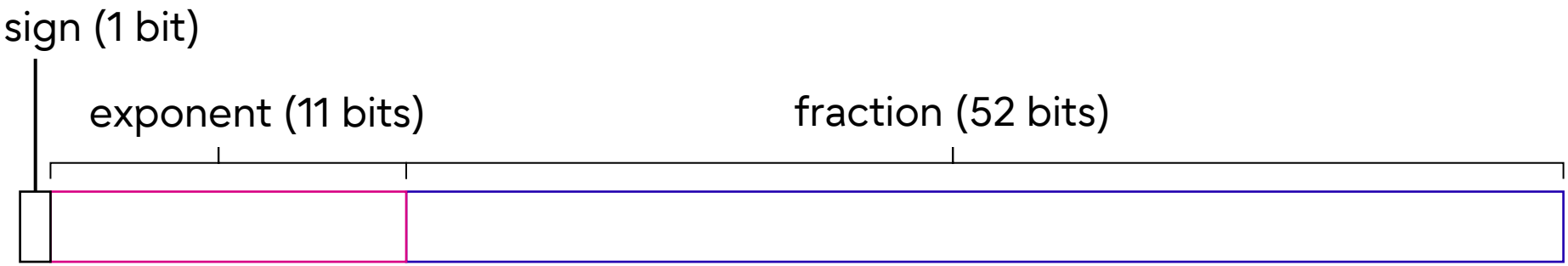
◆ kreal

integer(kind=4), parameter hecmw_util::kreal = 8

Definition at line **16** of file **hecmw_util_f.F90**.

Problem of Double precision FP number

- double-precision floating point numbers



- Representation: $(-1)^s \times (1 + \text{fraction}) \times 2^E$
- Exponent field size impacts the range.
- Fraction field size impacts the precision.
- When fraction part of computation results contains more than 52 bits, rounding occurs. → rounding error.

QPBLAS for rounding error reduction

- Quadruple Precision BLAS Routines (QPBLAS) has been developed by JAEA.
- QPBLAS provides high-precision linear algebra routines.
- QPBLAS simulates 128 bits computation using double-double algorithm (two 64-bits numbers represent one 128-bits number).
- QPBLAS leverages fast double-precision computation with almost the same precision of hardware-based 128 bits computation.

QPBLAS x FrontISTR

- Build FrontISTR with QPBLAS library
- CG with DD arithmetic and QPBLAS
- Data Structure modification
- Integration of DD arithmetic inside FrontISTR source code

Compilation of FrontISTR with DD library

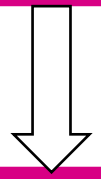
- We want to incorporate QPBLAS routine in FrontISTR for high-precision CAE simulation.

- Build phase: we add this line to the fistr1/CMakeLists.txt
  - target_link_libraries(fistr hecmw **-lblas -lddblas**)

Implementation of DD arithmetic in FrontISTR

- Implementation of DD arithmetic in **hecmw_dd** module
  - dd version of: add, sub, mul and div
- Integration of DD arithmetic for CG solver: we modified:
  - **hecmw_solve_CG**
  - solver_misc::hecmw_InnerProduct_R
  - solver_misc:: hecmw_xpay_R
  - solver_misc:: hecmw_axpy_R
  - solver_las::hecmw_matvec
    - hecmw_matvec_33, hecmw_matvec_33_inner
  - solver_las::hecmw_matresid
    - hecmw_matresid_33
  - hecmw_solver_scaling_33
  - hecmw_precond_apply (Diagonal Scaling preconditioner in DD has been implemented)
    - hecmw_precond_apply_33, hecmw_precond_DIAG_33

```
original_fistr_routine(X,Y,alpha)
```



```
dd_routine(XH,YH,alphaH,XL,YL,alphaL)
    real(kind=kreal), optional :: XL(:), YL(:), alphaL
```

Use of QPBLAS routines in FrontISTR

- Original hecmw code has been replaced by QPBLAS routines:
- **hecmw_solver_misc.f90**:
  - hecmw_innerproduct_R → **DDDOT**
  - hecmw_axpy_R → **DDAXPY**
  - hecmw_copy_R →**DDCOPY**

```
do i = 1, hecMESH%nn_internal * ndof
  Y(i) = X(i)
end do
```

```
if (present(XL)) then
  call DDCOPY(hecMESH%nn_internal*ndof, XH, XL, 1, YH, YL, 1)
```

Data structure modification

- CG solver of HECMW library relies on QPBLAS routine to perform high precision computation with DD algorithm.
  - CG solves Ax=b with DD variables: solve_lineq(hecMAT)
  - CG outputs xh and xl , the solution vector (= incremental displacement) into hecMAT%xh and hecMAT%xl new attributes.

```
type hecmwST_matrix
routine fstr_mat_init
routine hecMAT_init
```

hecMAT: new attributes

```
884 890        hecMAT%AL = 0.0d0
885 891        hecMAT%AU = 0.0d0
886 892        hecMAT%B  = 0.0d0
    893 +      hecMAT%BH  = 0.0d0
    894 +      hecMAT%BL  = 0.0d0
887 895        hecMAT%X  = 0.0d0
    896 +      hecMAT%XH = 0.0d0
    897 +      hecMAT%XL  = 0.0d0
888 898        hecMAT%ALU = 0.0d0
889 899      end subroutine hecMAT_init
```

hecMAT: new attributes: memory allocation

```
868 872        allocate (hecMAT%X(ndof*hecMAT%NP)          ,stat=ierror )
    873 +      allocate (hecMAT%XH(ndof*hecMAT%NP)          ,stat=ierror )
    874 +      allocate (hecMAT%XL(ndof*hecMAT%NP)          ,stat=ierror )
```

FrontISTR code: from x to disp results (1)

- type(hecmwST_matrix), pointer :: hecMATmpc
- For each timestep:
  - For each nonlinear iteration:
    - solution vector is stored into hecMATmpc%XH and hecMATmpc%XL
    - call solve_LINEQ(hecMESHmpc,hecMATmpc)
    - hecMATmpc%X[H/L] is copied into hecMAT%X[H/L]
    - call hecmw_mpc_tback_sol(hecMESH, hecMAT, hecMATmpc)
    - !the solution x (= incremental displacement) is added to the displacement value as an "update".
    - do j=1,hecMESH%n_node*ndof
      - fstrSOLID%dunode(j) = fstrSOLID%dunode(j)+hecMAT%X(j)
    - enddo

FrontISTR code: from x to disp results (2)

- For each timestep:
  - For each nonlinear iteration:
    - !updates the stress, strain and internal forces (not related to displacement value so not modified)
    - fstr_UpdateNewton(fstrSOLID, hecMAT)
    - !computation of new displacement, velocity and acceleration
    - do j = 1 ,ndof*nnod
      - fstrDYNAMIC%ACC (j,1:2) = -a1*fstrDYNAMIC%ACC(j,1) - a2*fstrDYNAMIC%VEL(j,1) + a3*fstrSOLID%dunode(j)
      - fstrDYNAMIC%VEL (j,1:2) = -b1*fstrDYNAMIC%ACC(j,1) - b2*fstrDYNAMIC%VEL(j,1) + b3*fstrSOLID%dunode(j)
      - fstrSOLID%unode (j) = fstrSOLID%unode(j) + fstrSOLID%dunode(j)
      - fstrDYNAMIC%DISP(j,2) = fstrSOLID%unode(j)

      enddo
    - !! Output the values

    call fstr_dynamic_Output(hecMESH, fstrSOLID, fstrDYNAMIC, fstrPARAM)

Final simulation results

Code snippet

```
!DD arithmetic
fstrSOLID%dunodeH = fstrSOLID%dunode
fstrSOLID%dunodeL = 0.0d0

do j=1,hecMESH%n_node*ndof
  !fstrSOLID%dunode(j) = fstrSOLID%dunode(j)+hecMAT%X(j) !original code
  call dd_add(fstrSOLID%dunodeH(j), fstrSOLID%dunodeL(j), hecMAT%XH(j), hecMAT%XL(j), ch, cl)
  fstrSOLID%dunodeH(j) = ch
  fstrSOLID%dunodeL(j) = cl
enddo

!DD Integration
fstrSOLID%dunode = fstrSOLID%dunodeH
```

Code snippet

```
! DD arithmetic
!fstrSOLID%unode(j)  = fstrSOLID%unode(j)+fstrSOLID%dunode(j) !original code
call dd_add(fstrSOLID%unodeH(j), fstrSOLID%unodeL(j), fstrSOLID%dunodeH(j), fstrSOLID%dunodeL(j), ch, cl
fstrSOLID%unodeH(j) = ch
fstrSOLID%unodeL(j) = cl

!fstrDYNAMIC%DISP(j,2) = fstrSOLID%unode(j) !original code
fstrDYNAMIC%DISPH(j,2) = fstrSOLID%unodeH(j)
fstrDYNAMIC%DISPL(j,2) = fstrSOLID%unodeL(j)
```

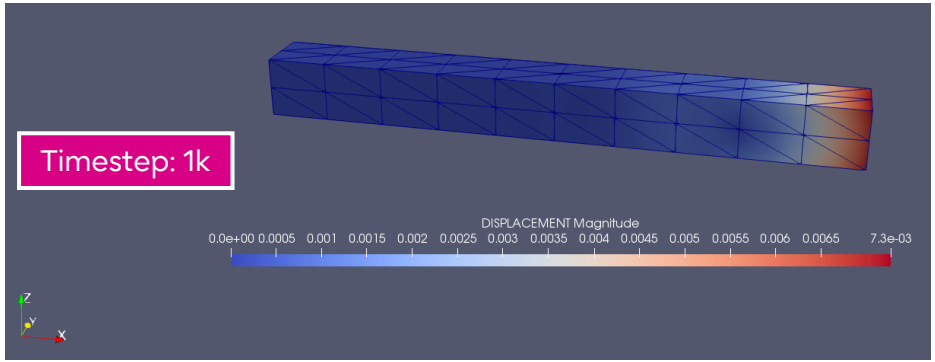
Evaluation

- Simulation results
- Numerical precision: number of digits
- Calculation time
- Conclusion
- PRC output

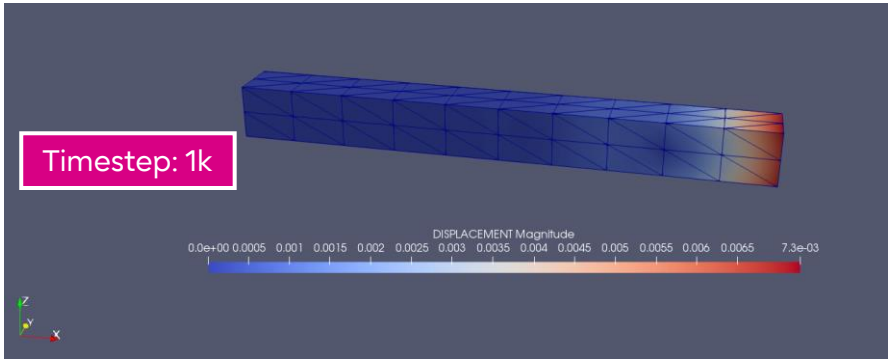
Comparison: simulation results

- We compare the original FrontISTR results with the one obtained with QPBLASxFrontISTR after 1000 timesteps analysis:

fistrDYNAMIC%DISP
computed from original hecMAT%X



fistrDYNAMIC%DISP
computed from hecMAT%XH



- Results are identical, showing that XH part is same as original X
- However, we also have XL part which increases the precision of XH

Comparison: numerical precision (digits number)

- We look at the numerical precision of:

Original FrontISTR
hecMAT%X

Last value of **hecMAT%X**:

1.047940573281168E-010

Double precision number:
16 digits

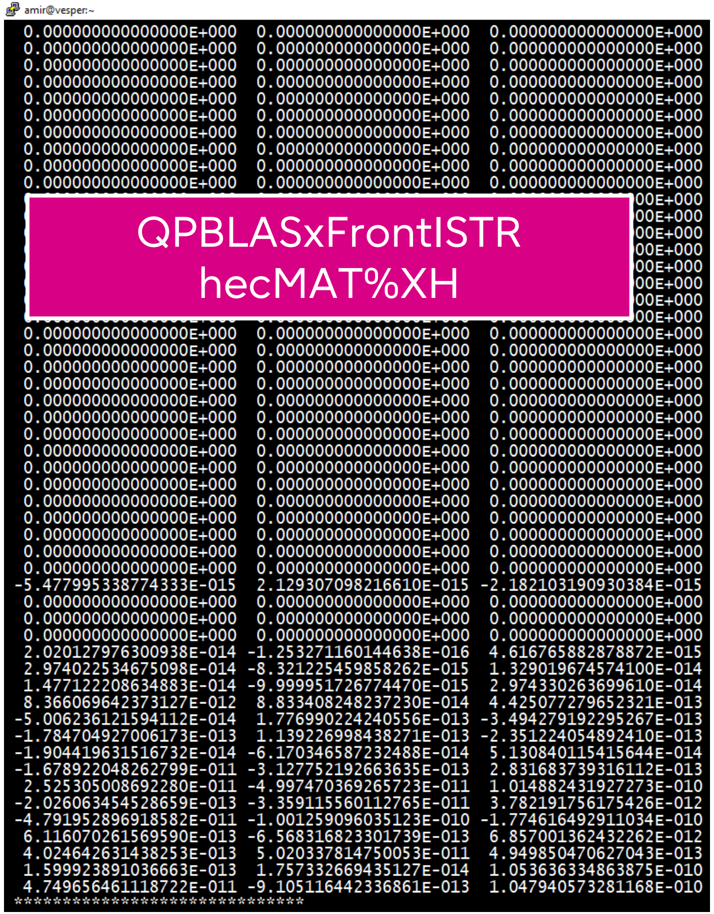
QPBLASxFrontISTR
hecMAT%XH
hecMAT%XL

Last value of **hecMAT%XH**:
1.047940573281168E-010
and corresponding **hecMAT%XL** value:
-2.351042639634128E-027

XH+XL= 1.047940573281167976489573603658 7E-10

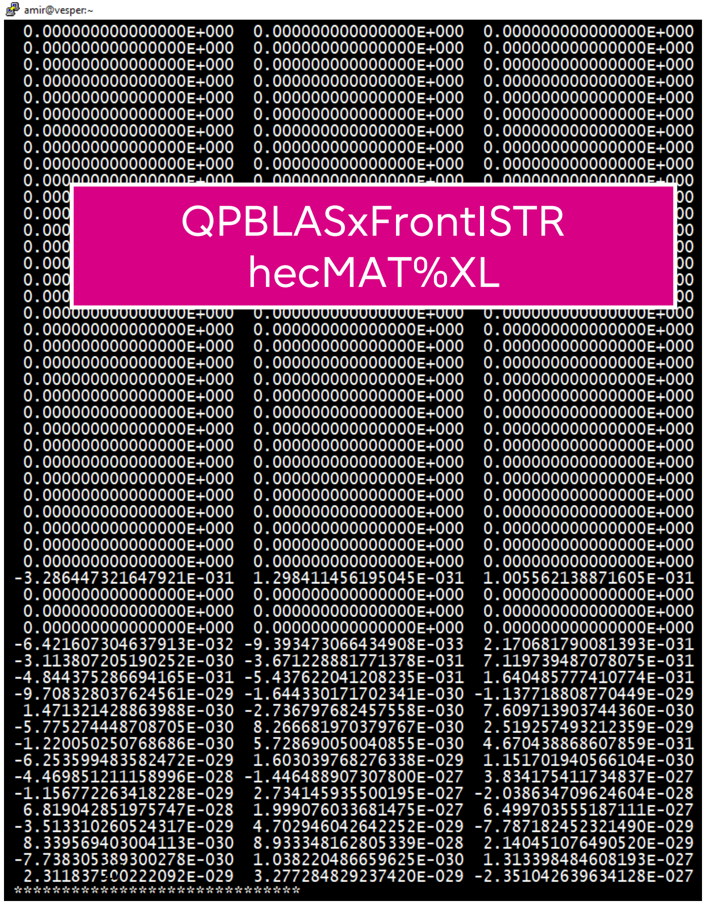
Quadruple precision number
32 digits ➔ rounding error is reduced

Results [details]



QPBLASxFrontISTR
hecMAT%XH

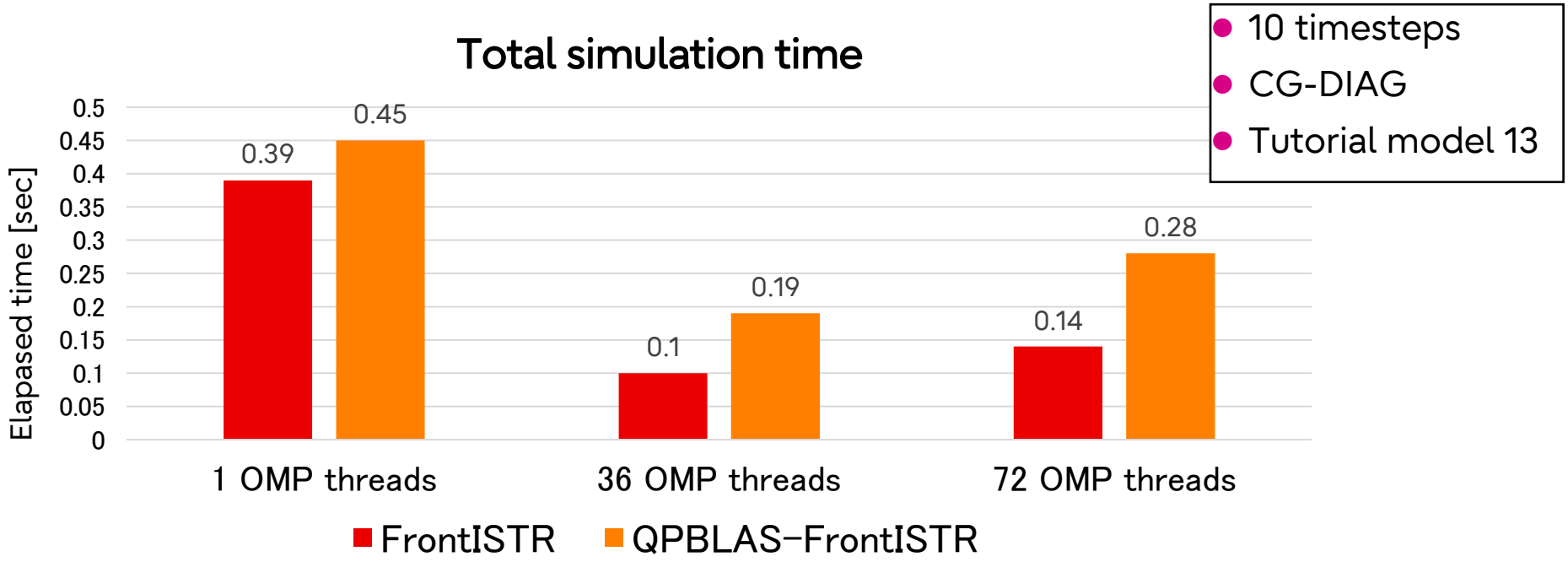
```
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
                                                                00E+000
                                                                00E+000
                                                                00E+000
                                                                00E+000
                                                                00E+000
                                                                00E+000
                                                                00E+000
                                                                00E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
-5.477995338774333E-015  2.129307098216610E-015 -2.182103190930384E-015
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 2.020127976300938E-014 -1.253271160144638E-016  4.616765882878872E-015
 2.974022534675098E-014 -8.321225459858262E-015  1.329019674574100E-014
 1.477122208634883E-014 -9.999951726774470E-015  2.974330263699610E-014
 8.366069642373127E-012  8.833408248237230E-014  4.425077279652321E-013
-5.006236121594112E-014  1.776990224240556E-013 -3.494279192295267E-013
-1.784704927006173E-013  1.139226998438271E-013 -2.351224054892410E-013
-1.904419631516732E-014 -6.170346587232488E-014  5.130840115415644E-014
-1.678922048262799E-011 -3.127752192663635E-013  2.831683739316112E-013
 2.525305008692280E-011 -4.997470369265723E-011  1.014882431927273E-010
-2.026063454528659E-013 -3.359115560112765E-011  3.782191756175426E-012
-4.791952896918582E-011 -1.001259096035123E-010 -1.774616492911034E-010
 6.116070261569590E-013 -6.568316823301739E-013  6.857001362432262E-012
 4.024642631438253E-013  5.020337814750053E-011  4.949850470627043E-013
 1.599923891036663E-013  1.757332669435127E-014  1.053636334863875E-010
 4.749656461118722E-011 -9.105116442336861E-013  1.047940573281168E-010
**********************************
```



QPBLASxFrontISTR
hecMAT%XL

```
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000                                                               00
 0.000                                                               00
 0.000                                                               00
 0.000                                                               00
 0.000                                                               00
 0.000                                                               00
 0.000                                                               00
 0.000                                                               00
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
-3.286447321647921E-031  1.298411456195045E-031  1.005562138871605E-031
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
-6.421607304637913E-032 -9.393473066434908E-033  2.170681790081393E-031
-3.113807205190252E-030 -3.671228881771378E-031  7.119739487078075E-031
-4.844375286694165E-031 -5.437622041208235E-031  1.640485777410774E-031
-9.708328037624561E-029 -1.644330171702341E-030 -1.137718808770449E-029
 1.471321428863988E-030 -2.736797682457558E-030  7.609713903744360E-030
-5.775274448708705E-030  8.266681970379767E-030  2.519257493212359E-029
-1.220050250768686E-030  5.728690050040855E-030  4.670438868607859E-031
-6.253599483582472E-029  1.603039768276338E-029  1.151701940566104E-030
-4.469851211158996E-028 -1.446488907307800E-027  3.834175411734837E-027
-1.156772263418228E-029  2.734145935500195E-027 -2.038634709624604E-028
 6.819042851975747E-028  1.999076033681475E-027  6.499703555187111E-027
-3.513310260524317E-029  4.702946042642252E-029 -7.787182452321490E-029
 8.339569403004113E-030  8.933348162805339E-028  2.140451076490520E-029
-7.738305389300278E-030  1.038220486659625E-030  1.313398484608193E-027
 2.311837500222092E-029  3.277284829237420E-029 -2.351042639634128E-027
**********************************
```

Computation time comparison



It looks like QPBLAS has some overhead compared with original FrontISTR. We want to investigate such results in future work. Todo: check solver iteration numbers, check how it evolves with increased problem size.

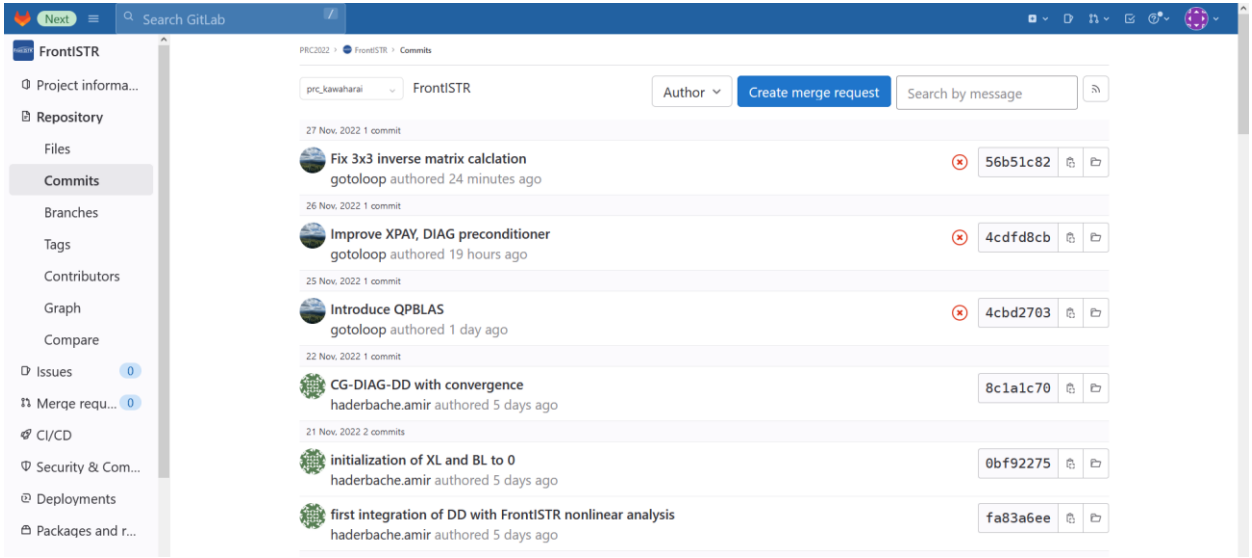
Conclusion

- We implemented and integrated QPBLAS calculation into FrontISTR FEM simulation in the scope of CG-DIAG solver.
- We confirmed that displacement results have higher numerical precision (32 digits versus original 16) with DD algorithm.

- Next work:
  - Find a simulation model which cannot converge with original FrontISTR CG-DIAG and shows that enhanced numerical precisision allows convergence for difficult problem. Try high condition number, poisson ratio=0.4999
  - Optimize QPBLASxFrontISTR computational time.
  - Implement I/O to output results files for DD values.

PRC Project: output

- QPBLAS X FrontISTR source code can be found at:
  - https://gitlab.com/prc2022/FrontISTR/-/tree/prc_kawaharai
  - https://gitlab.com/prc2022/FrontISTR/-/tree/prc_amir



Thank you very much for your time

Additional Materials

- DD Algorithm
- Interval Arithmetic

Linear system Ax=b

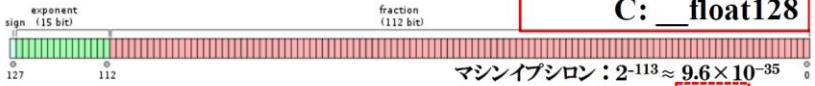
- A system of linear equation $\mathbf{Ax} = \mathbf{b}$ is solved, where:
  - $\mathbf{A} = \mathrm{M} + \mathrm{C}\gamma\Delta t + \mathrm{K}\beta\Delta t^2$ ➔ the "stiffness matrix"
  - $\mathbf{x} = \Delta\ddot{u}_{k+1}$ ➔ the increment of acceleration
  - $\mathbf{b} = \boldsymbol{\varepsilon} = f(t_{k+1}) - r(u_{k+1}, \dot{u}_{k+1}) - M\ddot{u}_{k+1}$ ➔ the nonlinear residual

- The **"stiffness matrix" A** is the sum of 3 sub-matrices:
  - The Mass matrix M
  - The Damping matrix C
  - The "linear" stiffness matrix K
  - γ and β are constant parameter of Newmark method

DD Algorithm

- 4倍精度[128bit, IEEE745形式]

Fortran: real*16
C: __float128

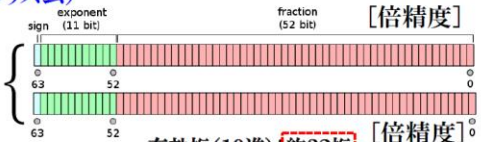


マシンイプシロン：$2^{-113} \approx 9.6 \times 10^{-35}$
有効桁（10進）約34桁

→ ソフトウェア的実装（低速）

- [擬似]4倍精度（DDアルゴリズム）
[64bit + 64bit]

[4倍精度]

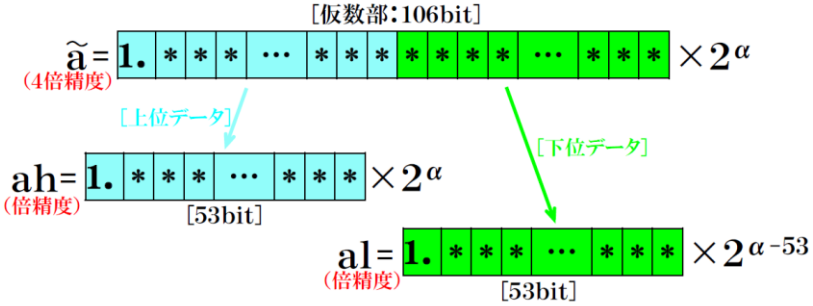


有効桁（10進）約32桁 [倍精度]

19/100

- DDアルゴリズム

※ 倍精度を2つ組み合わせて4倍精度を構成



34/100

$\tilde{a} = ah + al \rightarrow (ah, al)$
[上位] [下位]

FrontISTR code: from x to disp results (summary)

- call solve_LINEQ(hecMESHmpc,hecMATmpc)
- call hecmw_mpc_tback_sol(hecMAT, hecMATmpc)
- do j=1,hecMESH%n_node*ndof
  - fstrSOLID%dunode(j) = fstrSOLID%dunode(j)+hecMAT%X(j)
  - fstrSOLID%unode (j) = fstrSOLID%unode(j) + fstrSOLID%dunode(j)
  - fstrDYNAMIC%DISP(j,2) = fstrSOLID%unode(j)

- The final displacement results is into fstrDYNAMIC%DISP and is written into output res/vtk files