Benchmark tests towards GPU implementation of turbulence code

Masatoshi YAGI and Kenji Imadera

Acknowledgements

We acknowledge NVIDIA Japan for GPU implementation of GKNET code.

(I) Benchmark test of Global Gyrokinetic Code GKNET on GPU

(II) Benchmark test of Pseudo-spectral Code R5F on GPU

TESTING ENVIRONMENT/CONDITIONS

Machine: DGX station

- CPU: 1x Xeon(R) CPU E5-2698 v4
- GPU: 4x V100-DGXS-32GB

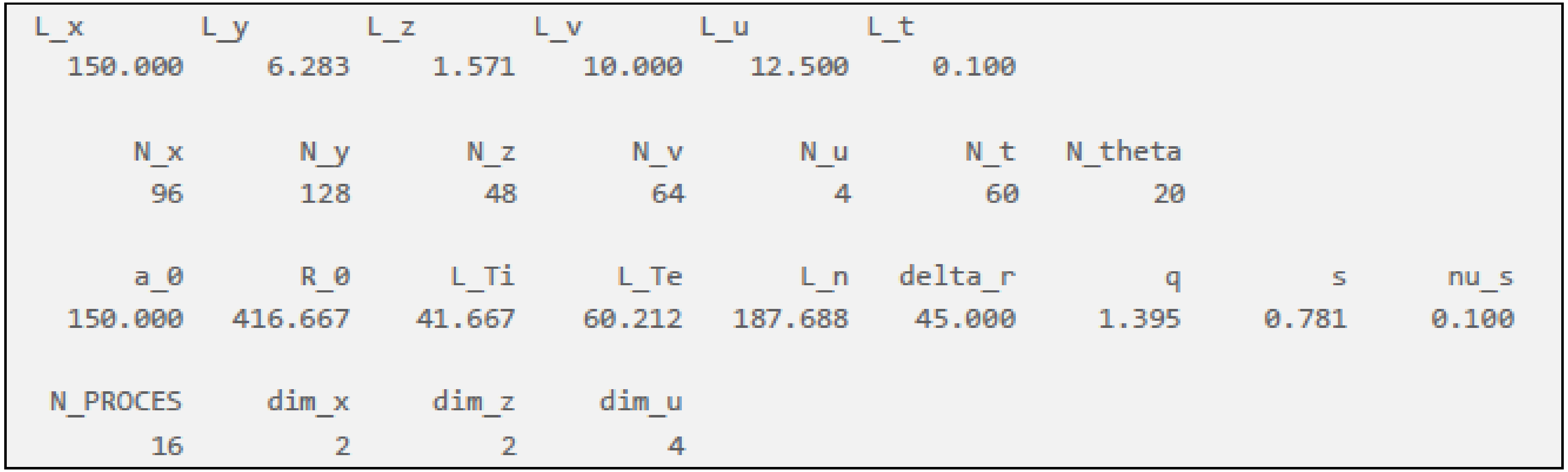
OS: Ubuntu 18.04.5 LTS

Compiler: NVIDIA HPC SDK 21.2

- OpenMPI: 3.1.5

P3DFFT: 2.7.9

FFTW: 3.3.8



```
L_x       L_y       L_z       L_v       L_u       L_t
  150.000     6.283     1.571    10.000    12.500     0.100

      N_x       N_y       N_z       N_v       N_u       N_t   N_theta
       96       128        48        64         4        60        20

      a_0       R_0      L_Ti      L_Te       L_n   delta_r         q         s      nu_s
  150.000   416.667    41.667    60.212   187.688    45.000     1.395     0.781     0.100

 N_PROCES     dim_x     dim_z     dim_u
       16         2         2         4
```

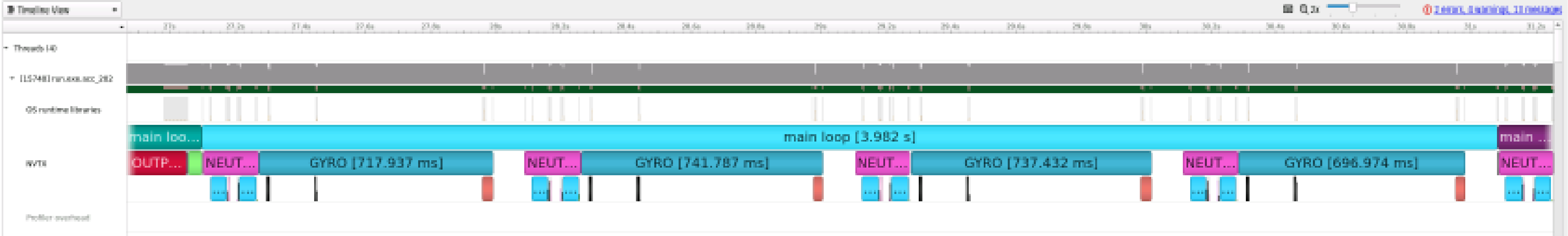
SINGLE CPU RUN

16 CPU-cores used

Single iteration time: 3.98 sec

- GYRO: 2.89 sec (73%)
- NEUTRAL: 0.67 sec (17%)

(*) when nothing is output



FOUR GPUS RUN

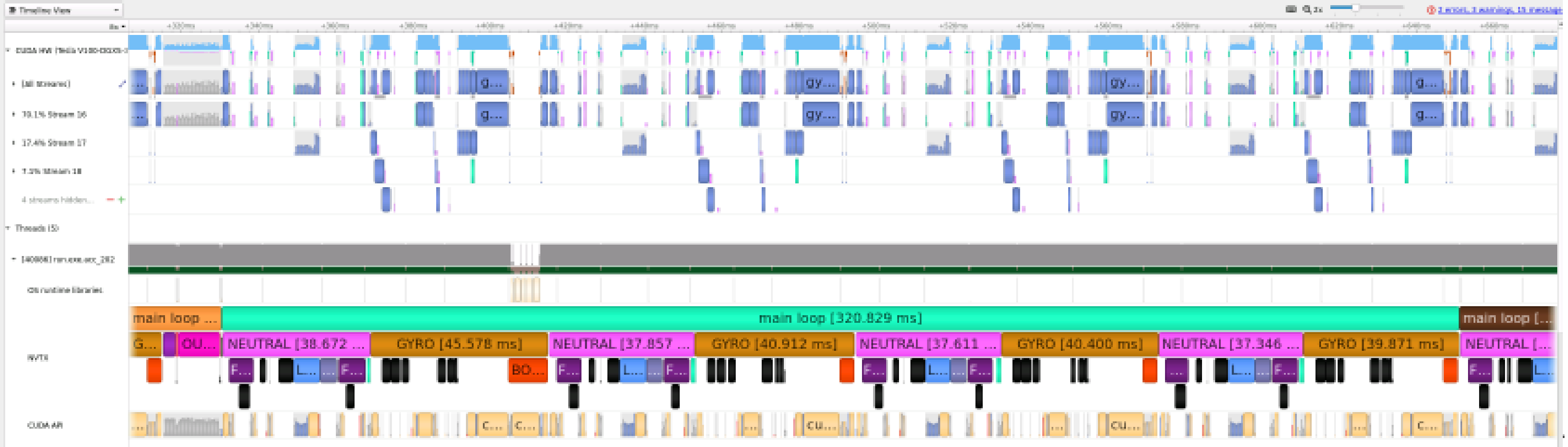
4 V100s and 16 CPU-cores used

Single iteration time: 320 ms

GYRO: 167 ms (52%)

NEUTRAL: 152 ms (47%)

(*) when nothing is output

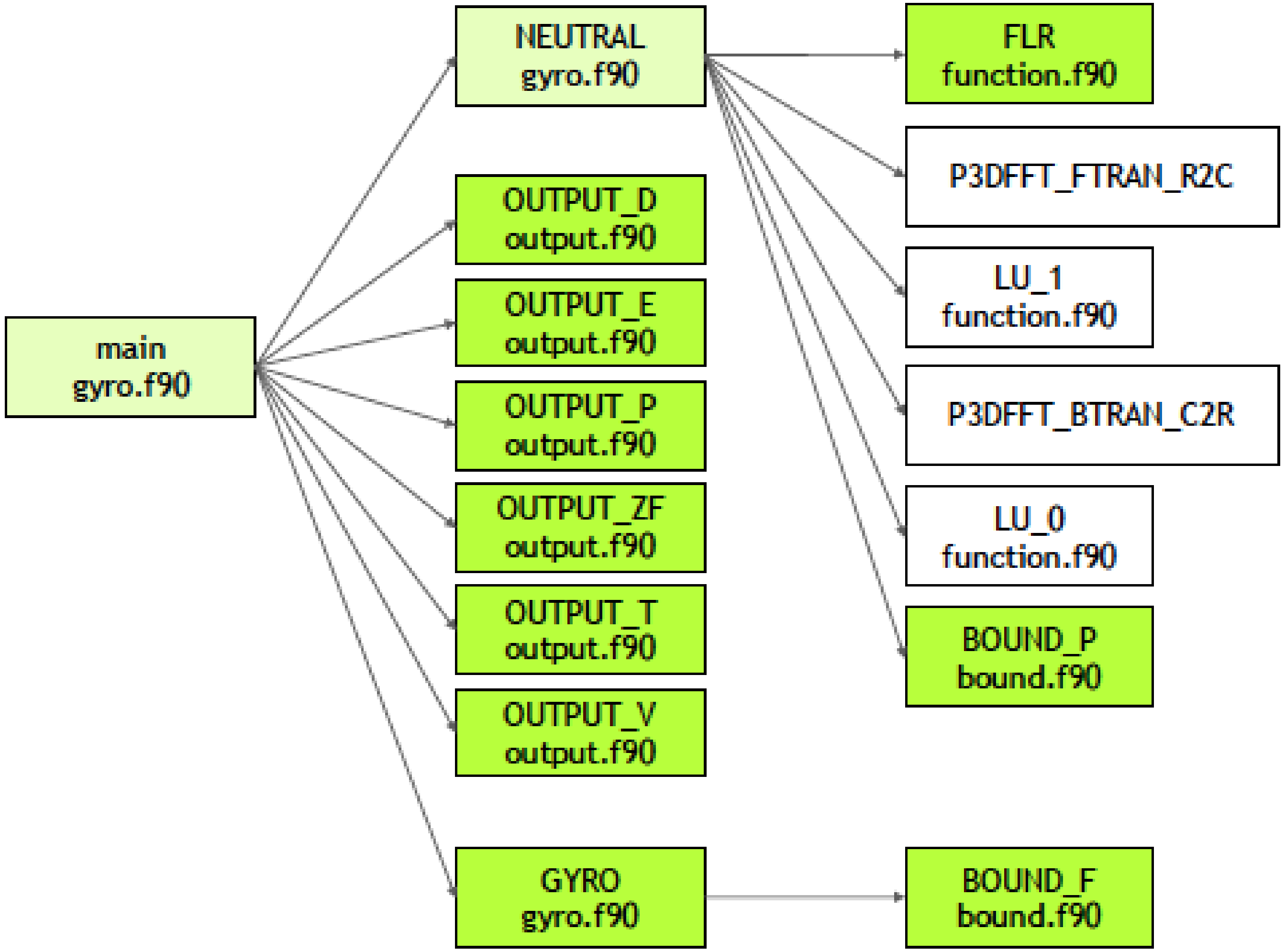


GPU ENABLED SUBROUTINES

GPU

Partially GPU

CPU



main gyro.f90

NEUTRAL gyro.f90

OUTPUT_D output.f90

OUTPUT_E output.f90

OUTPUT_P output.f90

OUTPUT_ZF output.f90

OUTPUT_T output.f90

OUTPUT_V output.f90

GYRO gyro.f90

FLR function.f90

P3DFFT_FTRAN_R2C

LU_1 function.f90

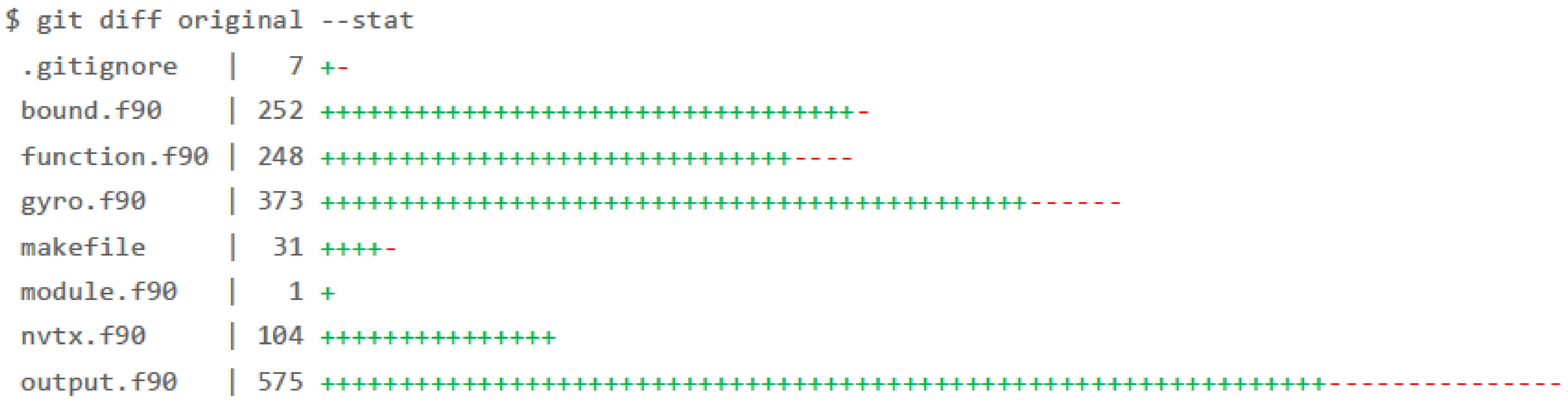
P3DFFT_BTRAN_C2R

LU_0 function.f90

BOUND_P bound.f90

BOUND_F bound.f90

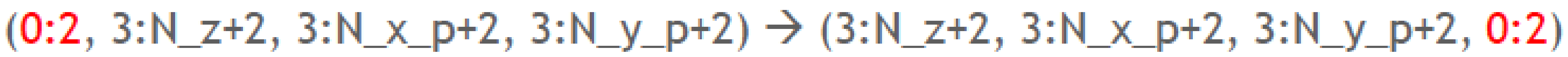
Modification of Source Code



```
$ git diff original --stat
 .gitignore    |   7 +-
 bound.f90     | 252 ++++++++++++++++++++++++++++++++++-
 function.f90  | 248 ++++++++++++++++++++++++++++++----
 gyro.f90      | 373 +++++++++++++++++++++++++++++++++++++++++++++------
 makefile      |  31 ++++-
 module.f90    |   1 +
 nvtx.f90      | 104 +++++++++++++++
 output.f90    | 575 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++---------------
```

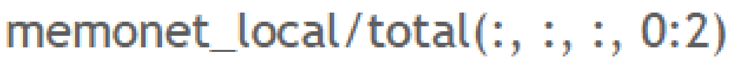
GYRO

To overlap calculations and communications in GYRO, shape of arrays moment_local/total are changed as



(0:2, 3:N_z+2, 3:N_x_p+2, 3:N_y_p+2) → (3:N_z+2, 3:N_x_p+2, 3:N_y_p+2, 0:2)

Calculations and communications of



memonet_local/total(:, :, :, 0:2)

are separated into 3 pieces.

```
!$acc wait
  !$acc kernels async(0)
  !$acc loop collapse(3) gang vector
  DO y_i = 3, N_y_p+2
   DO x_i = 3, N_x_p+2
    DO z_i = 3, N_z+2
     !$acc loop seq
     DO v_i = 4, N_v+3
      moment_local(z_i, x_i, y_i, 0) = moment_local(z_i, x_i, y_i, 0) + fs_i(v_i, z_i, x_i, y_i)*B_A(x_i, y_i)
     END DO
     moment_local(z_i, x_i, y_i, 0) = moment_local(z_i, x_i, y_i, 0) * w * d_v
    END DO
   END DO
  END DO
  !$acc end kernels
  !$acc update self(moment_local(:,:,:,0)) async(0)
  .
  .
  !$acc wait(0)
  call nvtxStartRange("AllReduce", 0)
  CALL MPI_ALLREDUCE(moment_local(:,:,:,0), moment_total(:,:,:,0), &
            N_x_p*N_y_p*N_z, MPI_double_precision, MPI_sum, MPI_comm_world_u, ierr)
```

BOUND_F

To use GPU direct, MPI subarray is changed to standard implementation.

1) Packing of data into communication buffer

2) MPI_isend, MPI_irecv are performed using communication buffer

3) Unpacking of data from communication buffer

```
#ifdef USE_GDR
    !$acc host_data use_device(f_send_w, f_recv_w)
#else
    !$acc update self(f_send_w)
#endif
    IF(xy_rank .LE. dim_xy(0)/2-1) THEN
      CALL MPI_IRECV(f_recv_w, 2*N_y_p*N_z*N_v, MPI_double_precision, a_rank, 10, MPI_comm_world_xy, ireq_a(1), ierr)
      CALL MPI_ISEND(f_send_w, 2*N_y_p*N_z*N_v, MPI_double_precision, a_rank, 20, MPI_comm_world_xy, ireq_a(2), ierr)
    ELSE
      CALL MPI_IRECV(f_recv_w, 2*N_y_p*N_z*N_v, MPI_double_precision, a_rank, 20, MPI_comm_world_xy, ireq_a(1), ierr)
      CALL MPI_ISEND(f_send_w, 2*N_y_p*N_z*N_v, MPI_double_precision, a_rank, 10, MPI_comm_world_xy, ireq_a(2), ierr)
    END IF


    CALL MPI_WAITALL(2, ireq_a, MPI_status_a, ierr)
#ifdef USE_GDR
    !$acc end host_data
#else
    !$acc update device(f_recv_w)
#endif
```

Benchmark test on M100

M100

CPU: IBM POWER9 AC922 (3.1GHz, 16cores) x 2

GPU: NVIDIA V100, Nvlink 2.0, 16GB x 4

```
$module load cuda/11.0
$module load hpc-sdk/2021--binary
```

Due to memory limitation, we use

N_x	N_y	N_z	N_v	N_u
64(96)	128	48	32(64)	4

CPU

TOTAL	COLLISION	VLASOV	BUFFER	SENDRECV	NEUTRAL	FLR	ALLRUDUCE	MATRIX	OUTPUT
1.181	0.208	0.416	0.010	0.012	0.020	0.333	0.001	0.037	0.134

GPU

TOTAL	COLLISION	VLASOV	BUFFER	SENDRECV	NEUTRAL	FLR	ALLRUDUCE	MATRIX	OUTPUT
0.510	0.097	0.089	0.023	0.037	0.036	0.068	0.003	0.092	0.060
	MPI_ALLtoALL			MPI_ISEND /IRECV	P3DFFT			LU	

Conclusion

GKNET has been implemented on GPU machine by NVIDIA Japan.

Total performance is ~2 times better than that on CPU.

GYRO part is ~4.7 times faster than that on CPU of M100.

Some part becomes slower on GPU, such as communication part and so on,
it should be improved in future, it is left for a future work.

Benchmark Environment

SGI8600	CPU: Intel Xeon Gold 624R (3.0GHz, 24cores) x 2
(JAEA/QST)	GPU: NVIDIA V100, Nvlink 2.0, 32GB x 4
JFRS-1 (Cray XC50)	CPU: Intel Xeon Gold 6148 (2.4GHz, 20cores) x 2
M100	CPU: IBM POWER9 AC922 (3.1GHz, 16cores) x 2
	GPU: NVIDIA V100, Nvlink 2.0, 16GB x 4

CPU Benchmark Results (SGI8600)

Intel Compiler

Compile Options:

```
-O3 –xCORE-AVX512 –fpp –I$(MKLROOT)/include/fftw
```

Link Options:

```
-O3 –xCORE-AVX512 -mkl
```

NVIDIA (pgi) Compiler

Compile Options:

```
-O4 –fast –mcmodel=medium –fastsse –Mpreprocess –I$(HOME)/fftw/include
```

Link Options:

```
-O4 –fast –mcmodel=medium –fastsse –L$(HOME)/fftw/lib64 –lblas -llapack –lfftw3
```

CPU Benchmark Results (SGI8600 Intel vs PGI)

Intel Compiler + MKL >> PGI Compiler + OpenBLAS + FFTW

Intel Compiler + MKL

	Routine	Process x Thread		
		32x1	64x1	128x1
	Total	3995.1	2163.1	1194.8
	Preprocess	3.2	3.7	3.0
Loop	MATRIX	0.46	0.24	0.12
	VECT	511.0	262.9	135.6
	TMRHS	1302.5	682.2	345.6
	TMPUS	1715.5	881.5	445.5
	GATHER	441.6	310.6	241.8

PGI Compiler + OpenBLAS + FFTW

	Routine	Process x Thread		
		32x1	64x1	128x1
	Total	12269.6	6315.9	3384.3
	Preprocess	3.0	2.3	3.2
Loop	MATRIX	0.47	0.23	0.12
	VECT	555.0	281.4	151.6
	TMRHS	1868.5	1007.5	514.8
	TMPUS	9302.5	4653.9	2266.0
	GATHER	501.9	332.5	409.1

Intel Compiler + MKL is roughly 3 times faster than PGI Compiler + OpenBLAS + FFTW!

CPU Benchmark Results (SGI8600 Intel vs JFRS-1 Cray)

Intel Compiler + MKL = Cray Compiler + MKL

Intel Compiler + MKL

	Routine	Process x Thread		
		32x1	64x1	128x1
	Total	3995.1	2163.1	1194.8
	Preprocess	3.2	3.7	3.0
Loop	MATRIX	0.46	0.24	0.12
	VECT	511.0	262.9	135.6
	TMRHS	1302.5	682.2	345.6
	TMPUS	1715.5	881.5	445.5
	GATHER	441.6	310.6	241.8

Cray Compiler + MKL

	Routine	Process x Thread		
		32x1	64x1	128x1
	Total	3856.8	2083.1	1201.9
	Preprocess	0.31	0.34	0.33
Loop	MATRIX	0.41	0.22	0.11
	VECT	460.6	236.0	126.4
	TMRHS	1262.4	662.9	357.7
	TMPUS	1756.2	880.7	445.7
	GATHER	356.8	281.7	249.6

Even though Intel Xeon Gold 624R > Intel Xeon Gold 6148, SGI8600 = Cray XC50

For 128 processes, 4xEDR (Hypercube) shows better performance than Aries interconnect (Dragonfly).

Code Analysis (SGI8600 Intel)

For 32 processes,

TMRHS:	1302.5	
	FFT	399.7
	Comm	63.9
	Calc	838.9

We may replace FFTW by CUFFT and apply OpenACC directives for loops in Calc.

TMPUS: 1715.5

call ZGBSV

Unfortunately, CUSOLVER does not support ZGBSV （Block-banded matrices), so that we will not apply GPU acceleration for TMPUS in this time.

Offload of FFT in TMRHS

FFTW

```
call dfftw_plan_dft_c2r_2d(&
    plan(1),NY,NZ,X1C,W1R,…)

DO I=S,E

    DO L=1,NDM

    W1C(IY,IZ)=DXVOK_T(I,L)

END DO

DO L=1,12

    call dfftw_execute(plan(L))

END DO
```

cuFFT

```
#ifdef CUFFT
use cudafor
use cufft
#endif
INTEGER(I4B) :: istat

istat=cufftPlan2d(cuplan(1),NZ,NY,..)

DO I=S,N
W1C_d=W1C

istat=cufftExecZ2D(cuplan(1),W1C_d,W1R_d)

W1R=W1R_d

END DO
```

MKL(CPU) vs cuFFT(GPU)

SGI8600 (MKL)

	Routine	Proc x Th
		32x1
	Total	3995.1
	Preprocess	3.2
Loop	MATRIX	0.46
	VECT	511.0
	TMRHS	1302.5
	TMPUS	1715.5
	GATHER	441.6

FFT: 399.7

SGI8600 (cuFFT)

	Routine	Proc x Th
		32x1
	Total	13893.4
	Preprocess	7.50
Loop	MATRIX	0.64
	VECT	654.3
	TMRHS	2848.4
	TMPUS	9747.2
	GATHER	597.6

FFT: 1800.2

M100(cuFFT)

	Routine	Proc x Th
		32x1
	Total	7618.7
	Preprocess	3.7
Loop	MATRIX	0.12
	VECT	486.3
	TMRHS	3192.7
	TMPUS	3450.7
	GATHER	463.8

FFT: 2677.4

cuFFT in M100 is ~1.5 times slower than that in SGI8600.

On the other hand, LAPACK (OpenBLAS) is ~2.8 times faster than that in SGI8600.

Calc in TMRHS (using OpenACC)

```
 !$acc kernels
copyin(S,E,KZMWD,KYMWD,IBAL,DXVOL_T,DYVOL_T,DXCUR_T,DYCUR_T,DXPHI_T,DYPHI_T,DXPSI_T,DYPSI_
T,DXVPL_T,DYVPL_T,DXPRE_T,DYPRE_T,DXDENS_T,DYDENS_T,DXTEME_T,DYTEME_T,DXTEMI_T,DYTEMI_T,COSTH,SIN
TH) create(I,N,M) copyout(VOLNON_T,PSINON_T,VPLNON_T,DENSNON_T,TEMENON_T)
    !$acc loop independent
    DO I=S,E
    DO N=1,KZMWD
    DO M=1,KYMWD
       VOLNON_T(M,N,I) = &
      -DXPHI_T(M,N,I)*DYVOL_T(M,N,I)+DYPHI_T(M,N,I)*DXVOL_T(M,N,I)  &
      +DXPSI_T(M,N,I)*DYCUR_T(M,N,I)-DYPSI_T(M,N,I)*DXCUR_T(M,N,I)
    .
    END DO
    END DO
    END DO
    .
  !$acc end kernels
```

```
mpif90 -O4 -fast -mcmodel=medium -fastsse -acc -Minfo=accel -Mcuda  -I/include -Mpreprocess -DCUFFT -DUSEMPI2 -c
tmrhs_5F.f90
tmrhs:
   292, Generating copyout(psinon_t(:,:,:)) [if not already present]
        Generating copyin(dyvpl_t(:,:,:),kymwd,ibal,e,dxtemi_t(:,:,:),dytemi_t(:,:,:)) [if not already present]
        Generating copyout(densnon_t(:,:,:)) [if not already present]
        Generating
copyin(dxvpl_t(:,:,:),dxdens_t(:,:,:),dydens_t(:,:,:),dxpre_t(:,:,:),dypre_t(:,:,:),dxvol_t(:,:,:),dyvol_t(:,:,:),dxphi_t(:,:,:),dyphi_t
(:,:,:),dxcur_t(:,:,:),dycur_t(:,:,:),dxpsi_t(:,:,:),dypsi_t(:,:,:),dxteme_t(:,:,:),dyteme_t(:,:,:),s) [if not already present]
        Generating copyout(temenon_t(:,:,:)) [if not already present]
        Generating copyin(kzmwd) [if not already present]
        Generating copyout(vplnon_t(:,:,:),volnon_t(:,:,:)) [if not already present]
        Generating copyin(costh(:),sinth(:)) [if not already present]
   295, Loop is parallelizable
   296, Loop is parallelizable
   297, Loop is parallelizable
        Generating Tesla code
       295, !$acc loop gang, vector(128) collapse(3) ! blockidx%x threadidx%x
       296,   ! blockidx%x threadidx%x auto-collapsed
       297,   ! blockidx%x threadidx%x auto-collapsed
       .
```

Calc (wo OpenACC): 331.8, Calc(w OpenACC) 587.2

It becomes worse!

Conclusion

Benchmark test of Pseudo-spectral code R5F is performed on GPU machine.

This code uses FFTW and LAPACK ZGBSV （Block-banded matrices) via MKL.

FFTW in TMRHS is replaced by cuFFT, however, the performance becomes worse.

DO LOOP is also parallelized using OpenACC, however, the performance becomes worse.

Masking the communication between CPU and GPU might be necessary to improve the performance.

In any case, it will take a time to optimize the code on GPU and it is left for a future work (especially, LAPACK functions should be).