Working week practical : miniapps

Cossevin Erwan, Penigaud Nicolas

Acdrag on the GPU

In the acdrag folder, it is possible to compare the speed of openaccsinglecolum / openaccmanyblocks on the Ampere GPU.

Openaccsinglecolumn is the former method of porting advised by NVIDIA (large kernels).

Openaccmanyblocks is the new method of porting, for more recent NVIDIA architectures.

To launch the miniapp :

- copy /home/soa1/practical_working_week
- source venv/bin/active and acdrag/env.gpu_nvhpc_d
- in acdrag/compile.gpu_nvhpc_d, make
- in acdrag, sbatch run_gpu.sh

Acdrag on the GPU

```
#!/bin/bash
#SBATCH -p gpu
#SBATCH --job-name=arp_acdrag
#SBATCH --nodes=1
#SBATCH --gres=gpu:4
#SBATCH --time 00:15:00

set -x

module load nvidia/24.5

SUBMIT_DIR=./acdrag_gpu.$$
mkdir $SUBMIT_DIR
cd $SUBMIT_DIR
arch=gpu_nvhpc_d

for method in openaccsinglecolumn openaccmanyblocks
do
../compile.${arch}/main_acdrag.x \
  --case-in /perm/soal/data/data_big \
  --verbose  --diff  \
  --nproma 32 \
  --ngpblks 2000 \
  --times 100 \
  --method $method > $method.txt 2>&1
done
```

In the launch script :

- the methods and input data directory need to be provided

- it is possible to set nproma (vector length), the number of blocks, the number of time the kernel is launched.

The output data (differences and execution time) is in the directory acdra_gpu.job_number, in a .txt file for each method.

ACDRAG on the GPU

The screen capture below shows the first lines of openaccsinglecolum.txt.

The computation time (ZCTTOT) and computation time per number of kernel runs and per number of grid points (ZCT) are given on the topmost line.

Those times are without data transfers, the corresponding times with data transfers are ZTDTOT and ZTD.

The lines below are the differences between the expected result (left) and the result computed on the GPU (middle).

By comparing with openaccmanyblocks.txt it can be checked that openaccmanyblocks is faster.

```
          CLARCH;                      CLMETHOD;  NPROMA; NGPBLKS;   NTIME;             ZTCTOT;                ZTC;             ZTDTOT;              ZTD
     cpu_nvhpc_d;           openaccsinglecolumn;      32;     2000;     100;    0.4037580490E+00;    0.6308719516E-07;    0.1050098896E+01;    0.1640779525E-06
 DIFF...
ZZSTRDU
   0.72118519506284050127E-04    0.72118519506284266968E-04   -0.21684043449710088680E-18      7     2     1
   0.71770407973595325934E-06    0.71770407973596818829E-06   -0.14928955695357043476E-19      8     2     1
  -0.15753292257013532921E-06   -0.15753292257013376750E-06   -0.15617169965001162502E-20     31     2     1
  -0.66724609780297346895E-07   -0.66724609780297280720E-07   -0.66174449004242213990E-22     32     2     1
   0.72215450107906755946E-04    0.72215450107906986339E-04   -0.23039296165316969223E-18      7     3     1
```

Bit-reproducibility for ACDRAG

The two folders acdrag_nobr and acdrag_br are prepared to see the bit-reproducibility possibilities of fxtran-acdc.

As explained, fxtran-acdc replaces the special functions (SIN, COS…) by bit-reproducible equivalents (and adds parentheses, but this is not available / needed here).

The scrips run_gpu.sh and run_cpu.sh in both directories runs the methods openmpsinglecolumn and openaccsinglecolumn.

In acdrag_nobr, using those scripts :
- it is possible to see the outputs of both methods is the same on CPU, but different on GPU.
- Also, both results on GPU are different from the result on CPU.

Bit-reproducibility for ACDRAG

Switching to the acdrag_br directory, the outcome is different : this time all results are equal.

The code of the transformed routine is available in :
compile.gpu_nvhpc_d/user-out/acdrag_openacc_bitrepro.F90, where it can be checked that the special functions have been replaced.

To do so, the following lines have been added to src/fxtran.conf :

```
'--use-bit-repro-intrinsics',
'--use-bit-repro-parens',
```

and the following directives have been added to src/acdrag.F90 (src : source code directory before transformation) :

```
!$ACDC singlecolumn
!$ACDC singlecolumn --suffix-singlecolumn _OPENACC_BITREPRO --use-bit-repro-intrinsics
!$ACDC manyblocks --max-statements-per-parallel=20
!$ACDC bitrepro
```

The call to ACDRAG_OPENACC_BITREPRO is added separately to main_acdrag.F90

Using streams in ACDRAG_MANYBLOCKS

Acdrag_manyblocks is composed of a dozen kernels.

It is possible to save a few percent of time by launching those kernels in an asynchronous stream.

To try the effect of this change, we can create a user-in directory in compile.gpu_nvhpc_d. This directory can also be used for debugging, for example.

If a generated file from user-out is modified and included in user-in, it will be used in the next compilation process.

We can therefore ask fxtran-acdc to create a new variant of manyblocks subroutine, ACDRAG_MANYBLOCKS_STR, and make the changes to the generated file.

```
!OPTIONS XOPT(NOEVAL)
SUBROUTINE ACDRAG (YDCST, YDML_PHY_MF,KIDIA,KFDIA,KLON,KTDIA,KLEV,&
 !-----------------------------------------------------------------------
 ! - INPUT  2D .
 & PAPRS,PAPRSF,PDELP,PNBVNO,PRDELP,PU,PV,&
 ! - INPUT  1D .
 & PRCORI,PGETRL,PGWDCS,PVRLAN,PVRLDI,&
 ! - OUTPUT 2D .
 & PSTRDU,PSTRDV,PRAPTRAJ)
!$ACDC singlecolumn
!$ACDC manyblocks --max-statements-per-parallel=20
!$ACDC manyblocks --suffix-manyblocks _MANYBLOCKS_STR --max-statement-per-parallel=20
```

Using streams in ACDRAG_MANYBLOCKS

To try the effect of streams, we add ASYNC(1) to all the ACC PARALLEL regions :

```
!$ACC PARALLEL LOOP GANG &
!$ACC&IF (LDACC) &
!$ACC&PRIVATE (JBLK) &
!$ACC&VECTOR_LENGTH (KLON) ASYNC(1)


DO JBLK = 1, KGPBLKS

  !$ACC LOOP VECTOR &
  !$ACC&PRIVATE (JLEV, JLON)
```

We also add a WAIT(1) at the end of the routine :

```
    ENDDO

  ENDDO

ENDDO
!$ACC WAIT(1)

IF (LHOOK) CALL DR_HOOK ('ACDRAG_MANYBLOCKS_STR', 1, ZHOOK_HANDLE)
!$ACC END DATA

!$ACC END DATA
```

With those changes, the kernels will be launched in order on the GPU, but the CPU will not wait for completion of a kernel before launching the next one. It will wait for the completion of all kernels launched in the ASYNC(1) at the WAIT(1).

Using streams in ACDRAG_MANYBLOCKS

By launching both acdrag_manyblocks and acdrag_manyblocks_str with run_gpu.sh, we can check that the second is a bit faster.

The reason is that the interval between kernels are removed, as seen below :