Porting LAM bi-Fourier transforms to GPU

Philippe Marguinaud, Thomas Burgot, CNRM/GMAP

April 2021

Table of contents

Introduction

This document describes the port of the bi-Fourier transforms used in AROME on GPU, following the work done by ECMWF on spherical harmonics transforms. In this document, the CPU is the host, and the GPU is the accelerator or device.

Spherical harmonics transforms

The spectral transforms library has been ported to GPU by NVIDIA and ECMWF. The code appears to be based on cycle 45. It is important to emphasize that **the code has been duplicated** and that there are now two versions of the code, one running on CPU, the other specialized for NVIDIA GPUs and using OpenACC directives.

The code was given in the following state :

- Arrays for computing on GPU are allocated in the setup, with a number of levels passed as argument to the setup routine, and the number of fields set to what IFS uses in its operational configuration. Apparently, allocating and deallocating four or five arrays for doing spectral transforms is a performance killer (dixit NVIDIA, confirmed by us).
- Commented code is found everywhere, although the code works.
- Managed mode is not used for metadata (dimensions, indirect access arrays, etc.); hence these data are replicated in arrays allocated on the GPU. Not using replicated metadata kills performance (dixit NVIDIA).
- CUDA aware transpositions are not coded entirely and would not be usable inside the model.
- A single resolution seems to be usable (ie it is not possible to switch from one resolution to another).
- cmake is used to build the code.

Software and hardware environment

Météo-France new HPC has been installed one year ago with several NVIDIA V100 nodes, but at the time of writing this report, we still do not have access to the PGI environment.

Therefore we had to use a dual NVIDIA GV100 node provided by ECMWF, without access to the debugger and the profiler. Let us recall that the GV100 is tailored for graphics, with a bandwidth of 700Gb/s instead of 900Gb/s for the V100. The PGI software is version 20.9, with OpenMPI and CUDA libraries (FFT, linear algebra, etc.)

We have configured and modified gmkpack to compile CUDA and use the PGI compiler.

A simple test

Let us assume that AROME uses a 4000x4000 grid with 100 levels. Let us assume that the number of fields involved in the spectral transforms is 10, and that the model runs using 400 MPI tasks.

It is easy to write a simple program that reproduces the FFT calculations a single MPI task would do in this particular configuration using CUDA FFTs.

Profiling such a simple test on the GV100 shows that :

- copying the data on the GPU takes about half a second (same for copying out), which gives us a PCI bandwidth of about 8Gb/s (this bandwidth is more likely to be 16Gb/s on our HPC).
- the computation itself takes about 40ms.

Bi-Fourier transforms make use of FFTs and transpositions; we can immediately see that FFTs will run fast but that moving the data will be very expensive.

Method used for porting

The work done by NVIDIA is described in this presentation:

https://www.ecmwf.int/sites/default/files/elibrary/2018/18628-accelerating-weather-prediction-nvidia-gpus.pdf .

We try to give more detail in this section.

This is an example of code, taken from the scalar version of `ltinv_ctl_mod.F90` :

```
!$OMP PARALLEL DO SCHEDULE(DYNAMIC,1) PRIVATE(JM,IM)
  DO JM=1,D%NUMP
    IM = D%MYMS(JM)
    CALL LTINV(IM,JM,KF_OUT_LT,KF_UV,KF_SCALARS,KF_SCDERS,ILEI2,IDIM1,&
     & PSPVOR,PSPDIV,PSPSCALAR ,&
     & PSPSC3A,PSPSC3B,PSPSC2 , &
     & KFLDPTRUV,KFLDPTRSC,FSPGL_PROC)
  ENDDO
!$OMP END PARALLEL DO
```

We see here that the dimension of this loop is `D%NUMP`, which is the number of wave numbers held by the current MPI task. This number will not be high enough to keep busy the thousands of cores present on a GPU (about 5k for a V100) if we try to use this distribution.

Furthermore, the operations performed by LTINV are complex and would not fit easily in the GPU framework (no NPROMA blocks here).

Therefore, it is more profitable to push this loop inside `LTINV`, and from `LTINV`, into all routines called by this routine; the final result is :

- for `ltinv_ctl_mod.F90` :

```
  CALL LTINV(KF_OUT_LT,KF_UV,KF_SCALARS,KF_SCDERS,ILEI2,IDIM1,&
   & PSPVOR,PSPDIV,PSPSCALAR ,&
   & PSPSC3A,PSPSC3B,PSPSC2 , &
   & KFLDPTRUV,KFLDPTRSC,FSPGL_PROC)
```

- for `leinv_mod.F90` :

We see here that loops are pushed near the calculations and collapsed, in order to increase parallelism; the following loop iterations will be distributed onto `D_NUMP` x `(R_NSMAX+3)/2` x `KFC` threads :

```
!$ACC PARALLEL LOOP COLLAPSE(3) private(KM,KDGLU,ISKIP,ISL)
DO KMLOC=1,D_NUMP
  DO J=1,(R_NSMAX+3)/2
    DO JK=1,KFC
      KM =  D_MYMS(KMLOC)
      IF(KM == 0)THEN
```

```
         ISKIP = 2
      ELSE
         ISKIP = 1
      ENDIF
      IF (MOD((JK-1),ISKIP) .EQ. 0) THEN
         ILA = (R_NSMAX-KM+2)/2
         IF (J .LE. ILA) THEN
            IA  = 1+MOD(R_NSMAX-KM+2,2)
            IZBA((JK-1)/ISKIP+1,J,KMLOC)=PIA(JK,IA+1+(J-1)*2,KMLOC)
         END IF
         ILS = (R_NSMAX-KM+3)/2
         IF (J .LE. ILS) THEN
            IS  = 1+MOD(R_NSMAX-KM+1,2)
            IZBS((JK-1)/ISKIP+1,J,KMLOC)=PIA(JK,IS+1+(J-1)*2,KMLOC)
         END IF
      END IF
    ENDDO
  ENDDO
ENDDO
```

This is a call to the CUDA `GEMM` routine; the other external function provided by NVIDIA/PGI environment is the FFT :

```
!$ACC host_data use_device(ZAA,IZBA,IZCAT,ZAS,IZBS,IZCST)
CALL CUDA_GEMM_BATCHED('N','T',ITDZCA,ILDZCA,ILDZBA,1.0_JPRBT,IZBA,ITDZBA,ILDZBA,&
      & ZAA,LDZAA,TDZAA,0._JPRBT,IZCAT,ITDZCA,ILDZCA,D_NUMP)
```

Note that the following idiom is used almost everywhere :

```
DO I = 1, N
  DO J = 1, NMAX (I)
    ...
  ENDDO
ENDDO
```

is transformed to :

```
JMAX = MAXVAL (NMAX)
!$acc parallel loop collapse (2)
DO I = 1, N
  DO J = 1, JMAX
    IF (J <= NMAX (I)) THEN
      ...
    ENDIF
  ENDDO
ENDDO
```

This has the effect to increase scalability, at the expense of losing computation power (cores not satisfying the inner condition waste cycles, waiting for others to complete their calculations). This is the first paradox of GPU: **more computing power is available, but it is harder to control it**.

Another side effect is that memory consumption increases; temporaries have to be allocated for all data to be processed, whereas in the CPU model, temporaries are allocated only for data being processed. This is the second paradox: **less memory is available on the GPU, but we need more**.

Dataflow

We take here the example of direct transforms.

```
SUBROUTINE DIR_TRANS(PSPVOR,PSPDIV,PSPSCALAR,PSPSC3A,PSPSC3B,PSPSC2,&
& LDLATLON,KPROMA,KVSETUV,KVSETSC,KRESOL,KVSETSC3A,KVSETSC3B,KVSETSC2,&
& PGP,PGPUV,PGP3A,PGP3B,PGP2)
```

Input arguments of `DIR_TRANS` (`PGP`, `PGPUV`, etc.) are expected to be located on the CPU. Output arguments (`PSPVOR`, `PSPDIV`, `PSPSCALAR`, etc.) will also be delivered on the CPU.

Direct transforms involve the following steps:

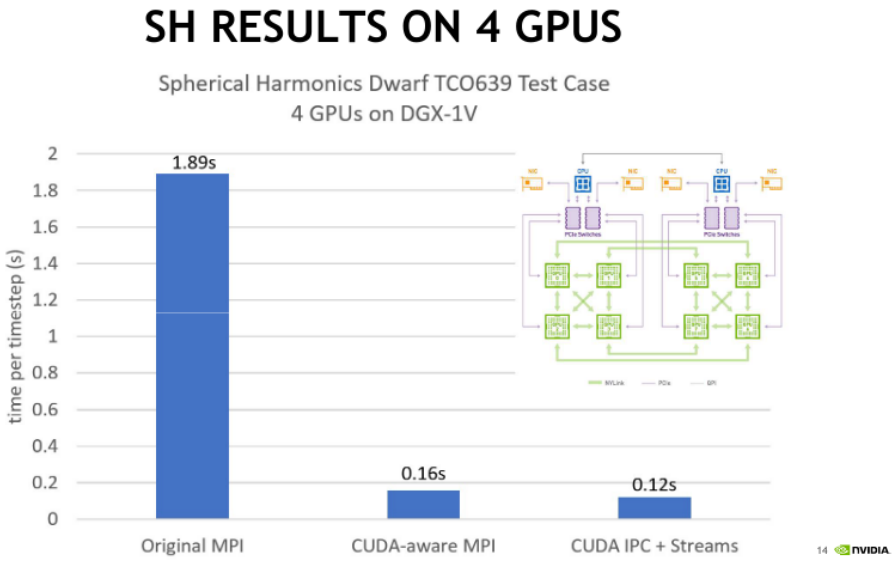
- TRGTOL transposition
- Direct Fourier transforms
- TRLTOM transposition
- Direct Legendre transforms

`TRGTOL` and `TRLTOG` have two versions : CUDA aware and non-CUDA aware. `TRLTOM` and `TRMTOL` should be CUDA aware, but the code is commented out. There is no real interest for `TRGTOL` (resp. `TRLTOG`) CUDA aware transpositions, as the input (resp. output) data for these transforms lies on the CPU: making them CUDA aware implies moving the data to the GPU before starting the communications, while using the non CUDA aware version makes the transposition and then moves the data to the accelerators. Maybe for some particular configuration using NVLink the CUDA aware `TRGTOL`/`TRLTOG` would be profitable.

The direct Fourier transforms following TRGTOL are performed entirely on the accelerator using CUDA FFT. All calculations (derivatives, vorticity/divergence, scaling, etc.) are also performed on the accelerator. It is of course highly desirable to avoid any data movement between the CPU host and the GPU.

`TRLTOM` and `TRMTOL` are probably meant to be CUDA aware, but again this would be profitable only by taking advantage of NVLink, which would add some constraints (we would need to make sure that these transforms are performed in a block of 8 GPUs). For now, `TRMTOL`/`TRLTOM` move their data to the host before doing the transposition.

The following diagram is taken from the NVIDIA presentation; it shows the dramatic improvement of performance when CUDA aware communications were enabled.



This configuration scales up to 8 GPUs only, using the dedicated NVLink intra-node network. Keeping this level of performance with 16 GPUs requires another hardware component called NVSwitch. Scaling beyond 16 GPUs would involve going through PCI, which is a well known bottleneck.

Another important lesson here is that the computation cost (FFTs and GEMMs) is very small compared to the cost of data movement. This is because **performing transpositions implies moving all data**.

Software modifications

We used the method described above for porting LAM transforms. We tried to use LAM features (`NDLON` is constant) as much as possible to improve the readability and performance of the code. In particular, Fourier transforms are not shared with the spherical transforms any more; this allowed us to run all FFTs in a single batch, which is not possible for spherical transforms (`NDLON` is not constant, different data layout). NVIDIA acknowledge that not being able to run these Fourier transforms in a single batch is a weakness.

This lead us to change the data layout of temporary arrays, and add an option to `TRGTOL`/`TRLTOG` transpositions so that this new data layout be taken into account. Arrays such as :

```
REAL(KIND=JPRB) :: ZGTF(KF_FS,D%NLENGTF)
REAL(KIND=JPRB) :: ZVODI(RALD%NDGLSUR+R%NNOEXTZG,MAX(4*KF_UV,1),D%NUMP)
```

are now declared :

```
REAL(KIND=JPRB)  :: ZGTF(D%NLENGTF,KF_FS)
REAL(KIND=JPRB)  :: ZVODI(RALD%NDGLSUR+R%NNOEXTZG,D%NUMP,MAX(4*KF_UV,1))
```

so that the number of field be the last dimension. All loops involving these arrays have been updated accordingly.

We also have introduced an option to **keep the memory allocated on the accelerator**, while trying to re-use it as much as possible; **the gain is about 30%**. This is because once freed, memory allocated by the current process is entirely returned to the OS; allocating/deallocating memory causes page faults, and the OS takes some time to handle them.

This looks like the third GPU paradox: we have very little memory, we need more, and we cannot deallocate it. This is of course not acceptable. But this situation could probably be solved using a custom allocator, or asking NVIDIA to provide an allocator which does not return GPU dynamic memory to the OS.

Eventually, note that having the number of fields as the last dimension of work arrays allowed us to grow these arrays when necessary (eg inverse transforms involve more fields than direct transforms). This currently not possible in the global transforms library, where arrays are allocated in the setup and not meant to be extended.

Managed mode

We tried managed mode and to our great surprise, it brings an improvement of about 20% on elapsed time (NVIDIA claims it would kill performance). It is therefore very likely that we have some data transfers happening without being noticed.

Currently, we do not have access to profilers and debuggers, so we cannot solve this issue.

We would also like to see whether managed mode could help us avoid metadata (dimensions, small arrays of indices) replication on the GPU.

Validation

We have written two test programs :

- `aatestprogder.F90` computes direct and inverse transforms of a list of scalar fields and their derivatives
- `aatestproguv.F90` computes direct and inverse transforms of a list of U/V fields

These two programs have been tested on a 20x20 grid, using pure harmonics produced in grid-point format by the CPU version. We have checked that CPU harmonics are also GPU harmonics, that is that the transform of such a grid-point field yields a spectral fields with a single non zero coefficient, and that this coefficient value is one.

Wind fields direct and inverse transforms have been tested using a 1000x1000 grid.

Derivatives have been checked on a few pure harmonics and on an analytic field $(X(1-X)Y(1-Y))^2$ on a 1000x1000 grid.

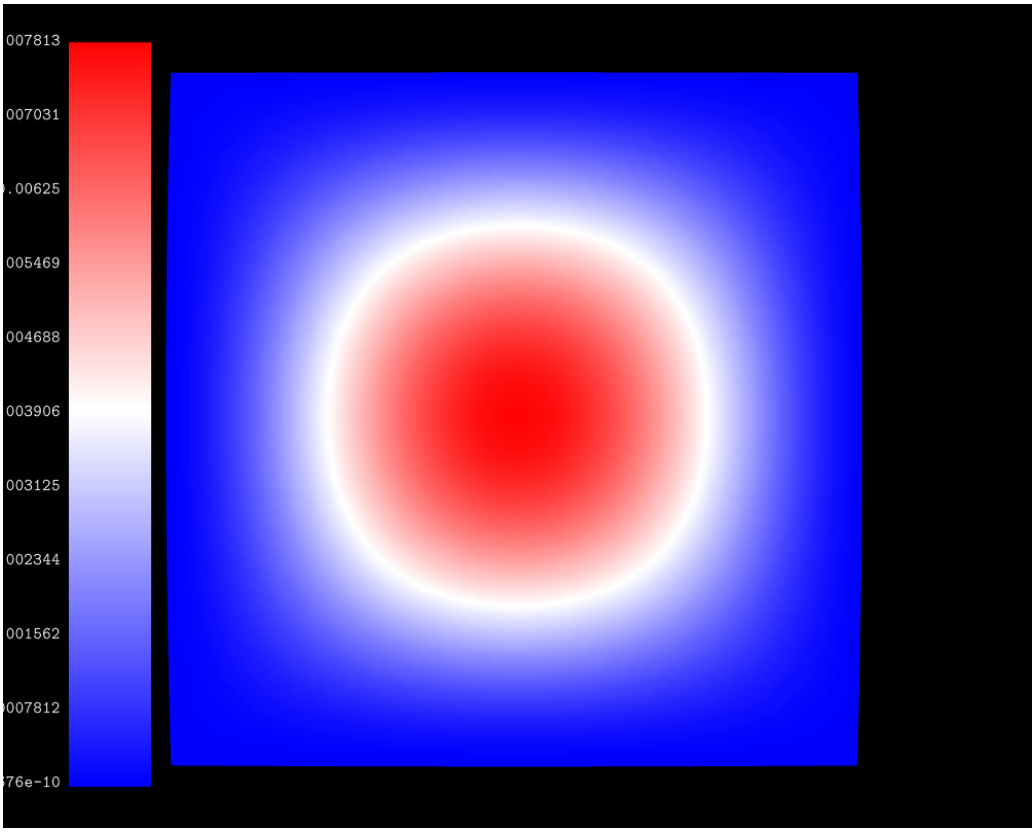


Figure 1: Analytic scalar field (X(1-X)X(1-X)Y(1-Y)Y(1-Y))

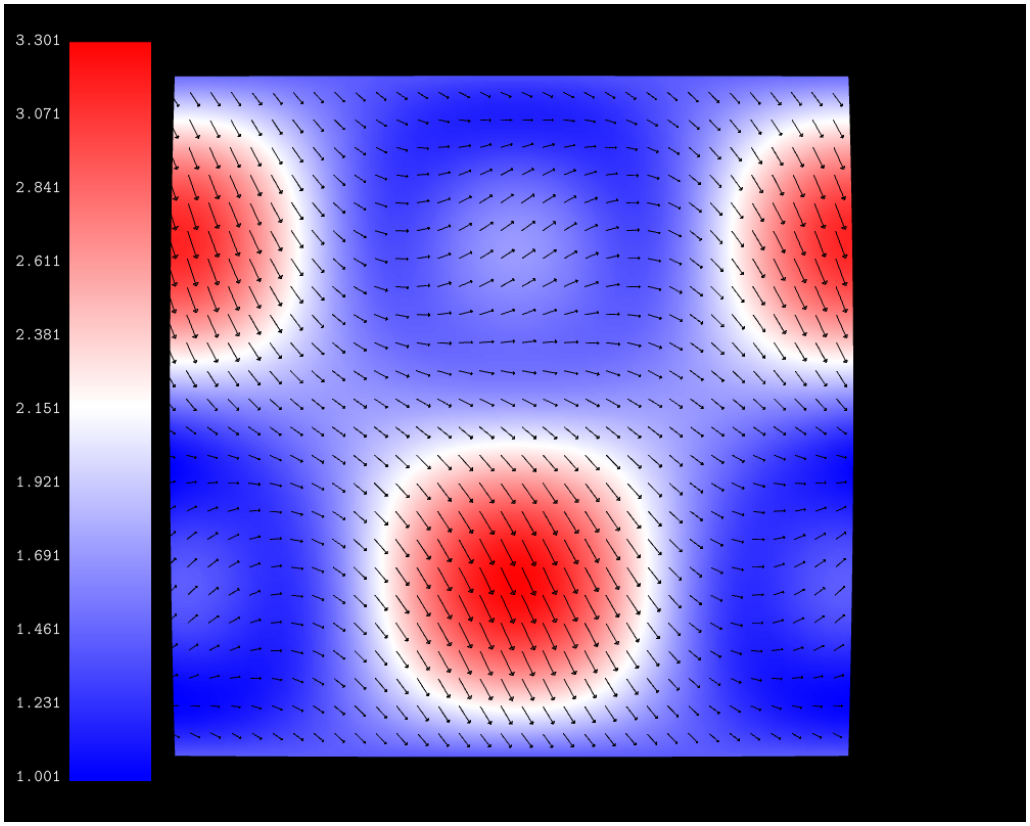


Figure 2: Analytic wind field on a 1000x1000 grid

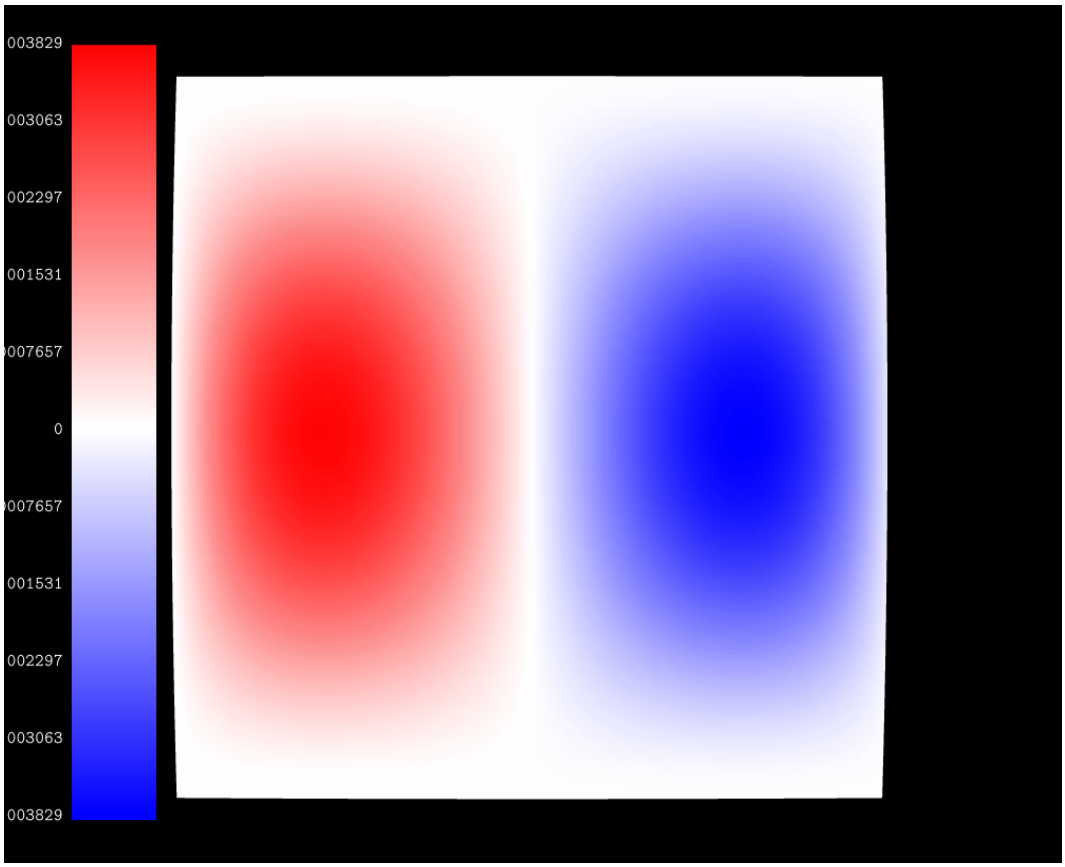


Figure 3: X derivative of (X(1-X)X(1-X)Y(1-Y)Y(1-Y))

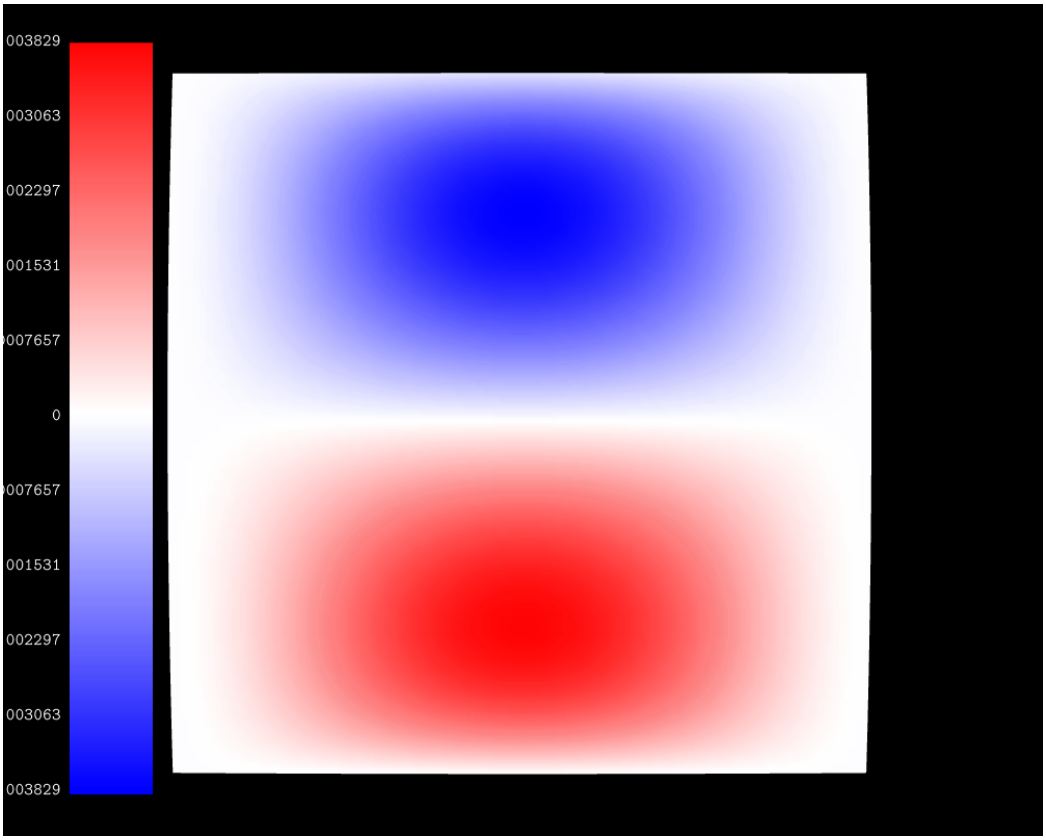


Figure 4: Y derivative of (X(1-X)X(1-X)Y(1-Y)Y(1-Y))

Conclusion and perspectives

LAM bi-Fourier work on NVIDIA GPUs. Quite a lot of work remains :

- Bi-periodicisation (is it still required ?) has not been coded.
- Adjoint transforms have not been coded.
- U/V derivatives have not been tested.
- PGP3A, PGP3B, PGP2 arguments have not been tested.
- Simple precision has not been tested.
- Our application has not been profiled.
- Managed mode has not been explored.
- Memory should be deallocated.