

*PAON4/IDROGEN*

---

# Corrélateur PAON4 TACQ / GPU

---

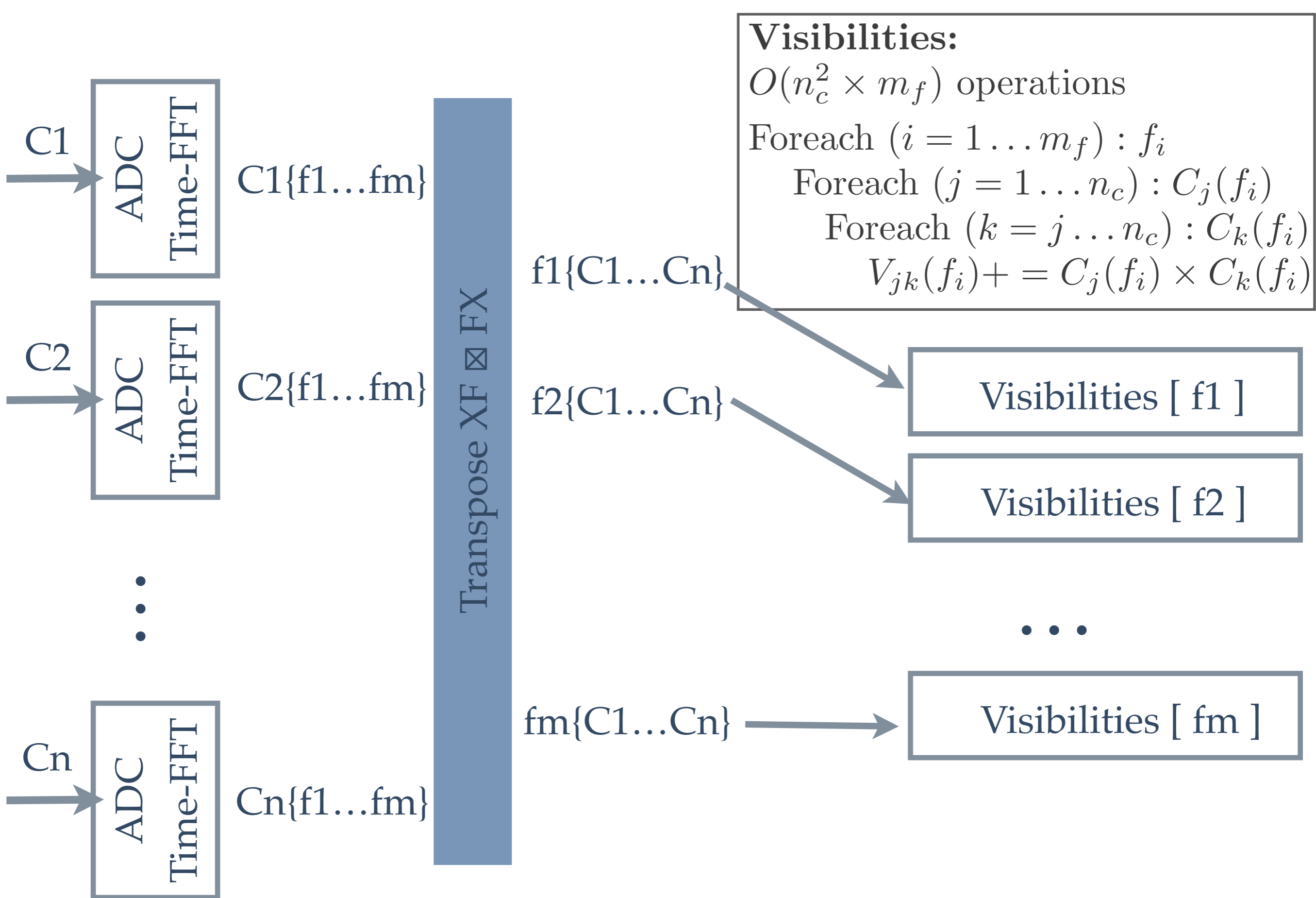
R. Ansari

Juillet 2020

Updated : December 2020

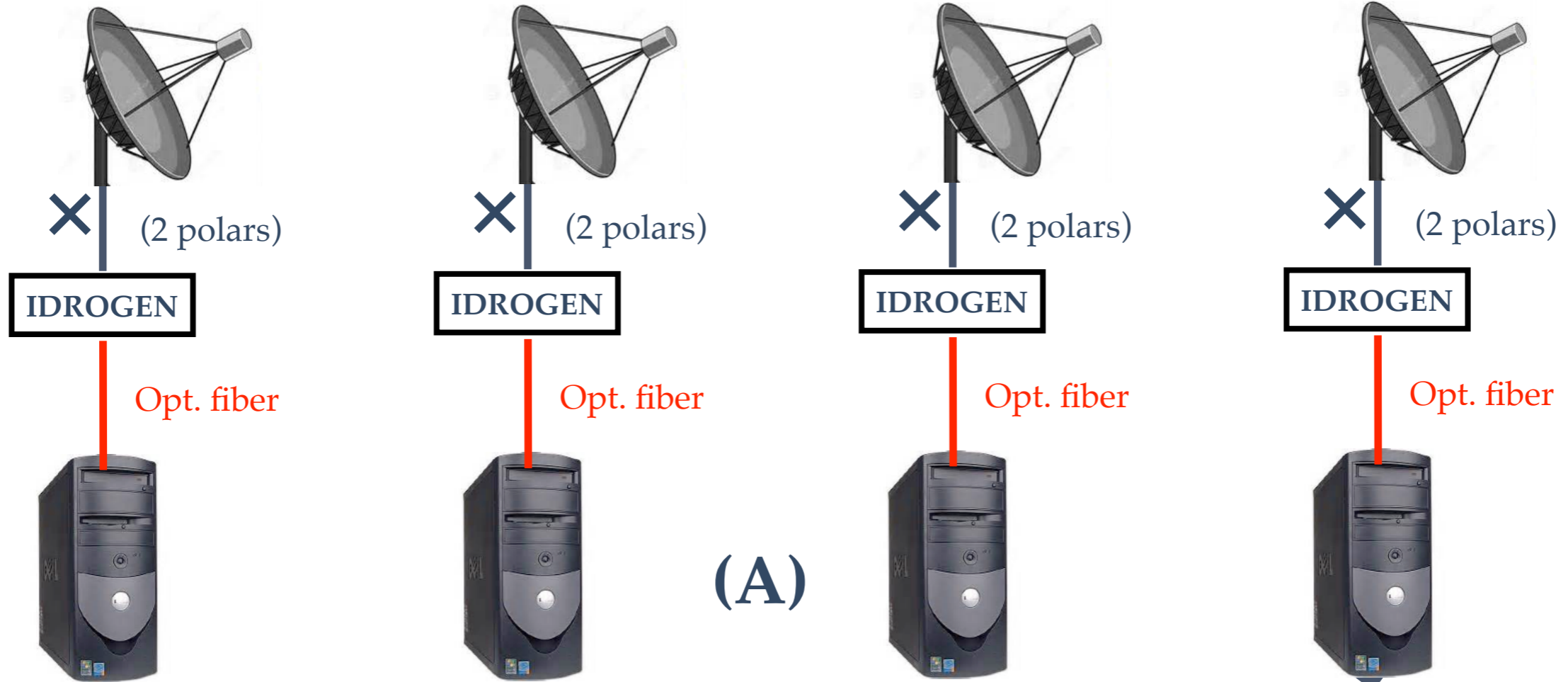


Configuration système d'acquisition  
Data rate update to be done ...

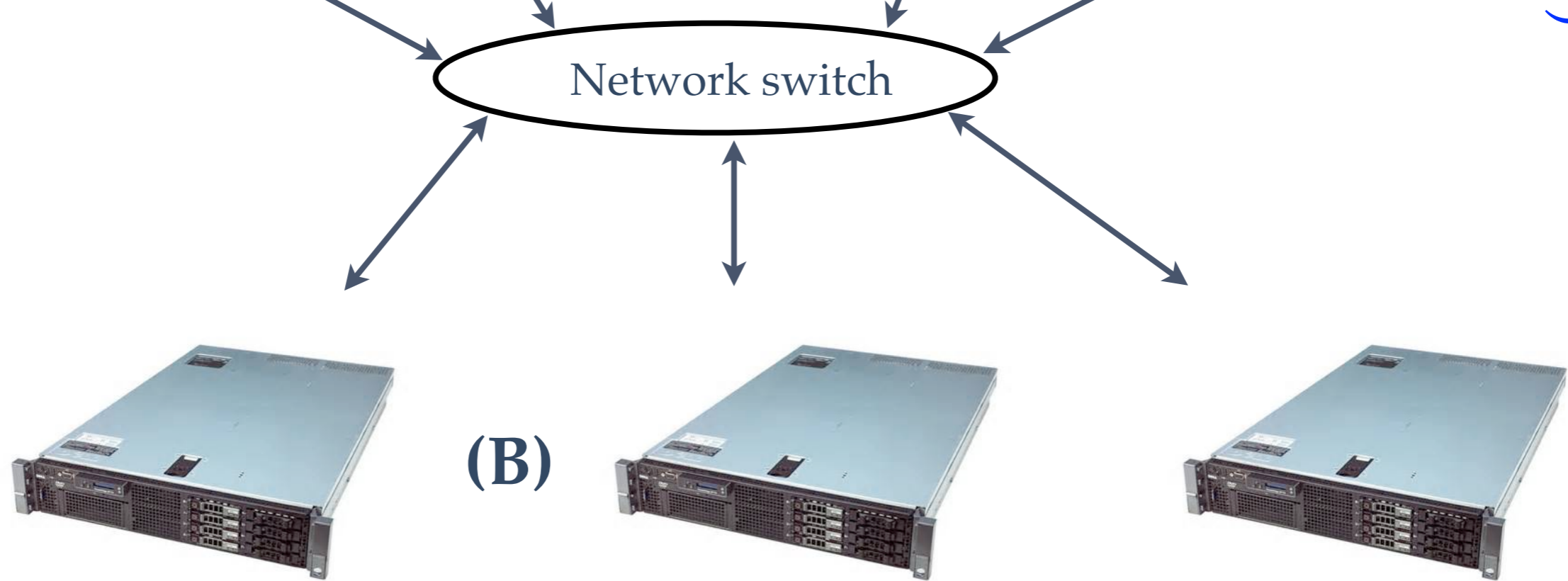




**(B) Visibility  
computing nodes**



**(A) Data reception/  
dispatch nodes**

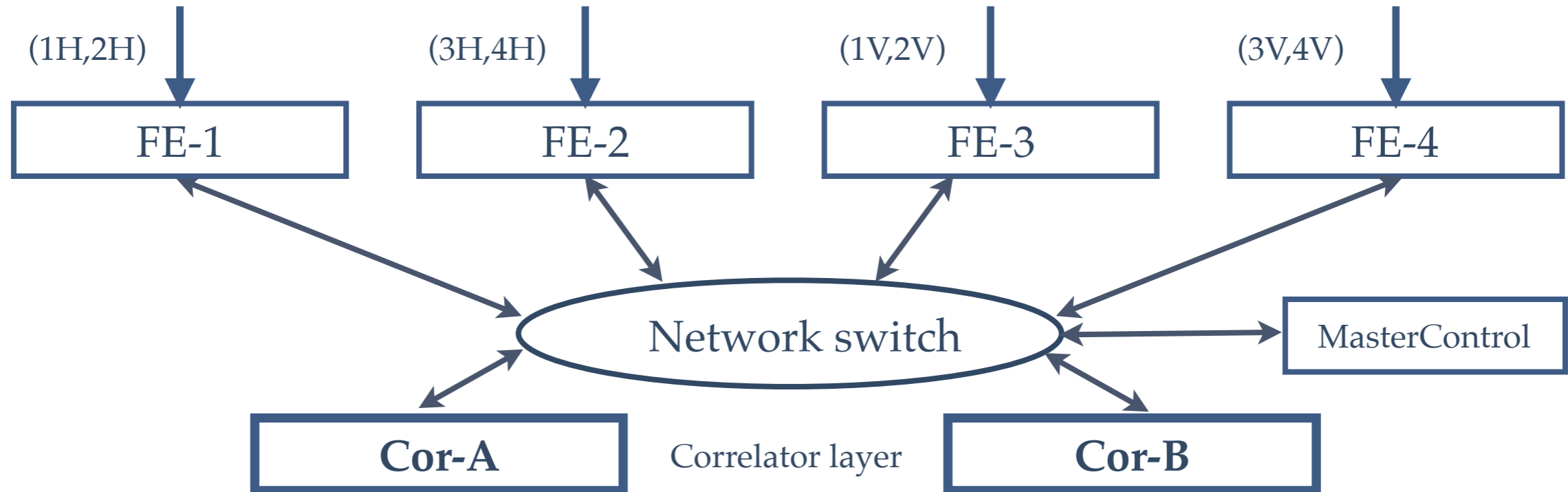


## **mfacq -pci2eth**

datacards:

```
@pci2ethsplit N_Bands
```

```
@pci2ethtargets B-node-1 B-node-2 ...
```



## **mfacq -eth2visib**

datacards:

```
@ethrnlink 4
```

```
@ethrsplitband 0 2
```

```
@ethrforcesamefc 0 0
```

## **mfacq -eth2visib**

datacards:

```
@ethrnlink 4
```

```
@ethrsplitband 1 2
```

```
@ethrforcesamefc 0 0
```



# Rappels

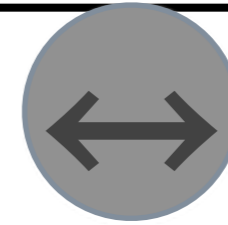
(TAcq : RACqMemZoneMgr ,  
BRVisibilityCalculator, BRPAquet ...)

<https://gitlab.in2p3.fr/baoradio/tacq>

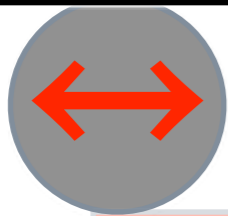
<https://gitlab.in2p3.fr/baoradio/tacq/-/wikis/home>

Acquisition/visibility  
computation (mfacq)

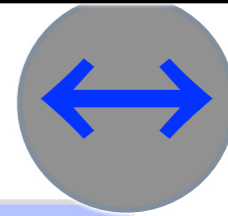
**ZThread**



**T1 - ReadEthernet**



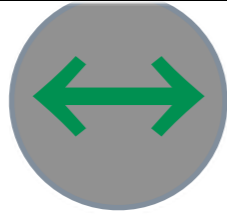
**BRVisibilityCalculator**




P11	P12	P13	P14	...		
P21	P22	P23	...			
P31	P32	...				

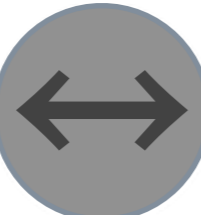
**RAcqMemZoneMgr**

**T3 - Monitoring**



Threads : EthernetReader,  
VisiCalc, Monitoring ...  
Task: PCIExpress  Ethernet

**PCIExpress**



**DMA-Task**



**Ethernet**

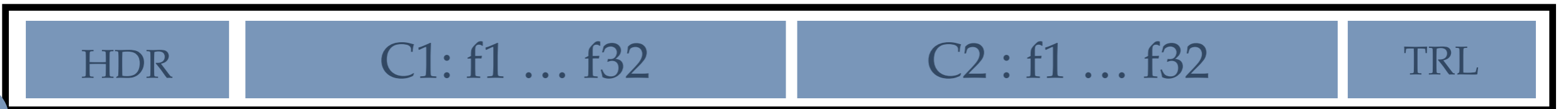
# BRPaquet



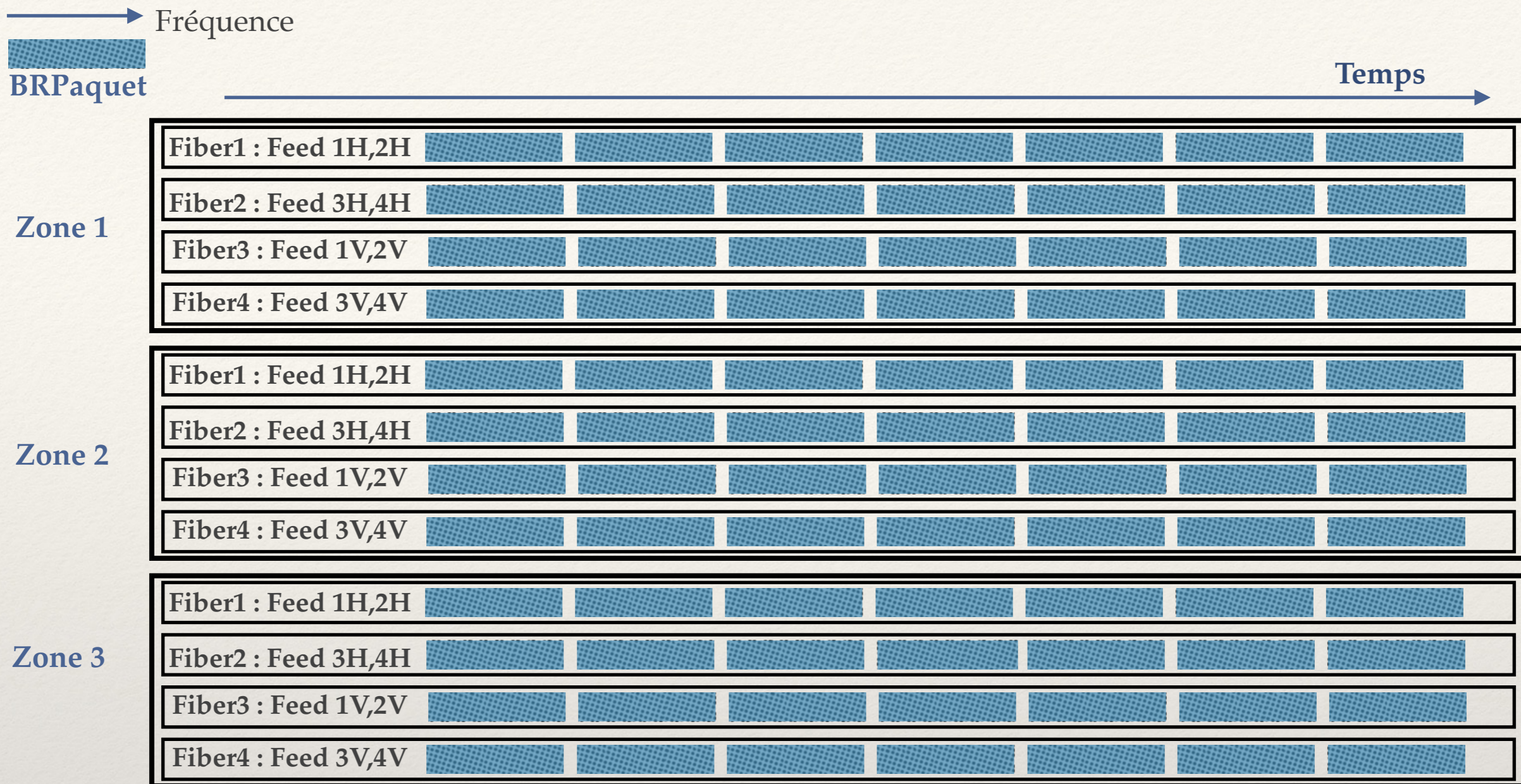
- HDR/TRL: Data description, HW Time Tag, paquet length ...
- Data : Raw or FFT coefficients, one or multiple channels
- HDR+TRL = 40 Bytes, Paquet length (variable)

HDR :

- Packet length
- DataDesc
- TimeTag
- FrameCounter







RAcqMemZoneMgr

Calcul de corrélation sur une ou plusieurs zones -> une matrice de visibilité





## Calcul des corrélations

Classes `BRVisibilityCalculator` et `BRVisCalcGroup`

méthode `BRVisibilityCalculator::execute(int tid)`,

~120 lignes sur les 1200 lignes de code de [brviscalc.cc](http://brviscalc.cc) ,

et encore, `execute()` traite différents cas (BRPaquet contenant une ou deux voies, Firmware FFT ou FFT soft ...) - la triple boucle fait environ 30 lignes



---

# BRVisibilityCalculator

---

- ❖ Deux niveaux de parallélisme implémentés : calcul parallèle sur les fréquences dans chaque objet BRVisibilityCalculator , parallélisme au niveau des paires (baselines) implémentés en lançant plusieurs BRVisibilityCalculator, gérés par BrVisCalcGroup
- ❖ Implémentation parallèle avec des threads
- ❖ Implémenter une nouvelle version, avec les deux niveaux de parallélisme implémentés dans la classe BRVisibilityCalculator pour les nouvelles machines avec pas mal d'unités de traitement (cores)



# Programme de test GPU/OpenCL

SophyaProgs/Eval/DivTst/tstopencil.cc

fonction:

```
void test_correl(cl::Device & default_device, size_t maxwksz0)
```



*kernel - noyau openCL  
comprend la boucle sur les  
fréquences*

```
cout << "[2] Creating kernels ..." << endl;

cl::Context context(default_device);
cl::Program::Sources sources;
// kernel calculates correlations (C_ij[freq] +=
std::string kernel_code=
"    void kernel rz_correl(global const float* A, global float* C,          "
"                          ulong nfeed, ulong nfreq, ulong nfw, ulong wkgsz)  "
"    {                                                                    "
"        size_t numvis = get_group_id(0);                                "      GroupID = num de visibilité
"        size_t kfreqoff = get_local_id(0)*(size_t)nfw*2;                "      LocalId = numéro de paquet
"        /* size_t ifeed = numvis/nfeed; size_t jfeed = numvis%nfeed; */  "
"        size_t ioff = (numvis/nfeed)*nfreq*2;                            "
"        size_t joff = (numvis%nfeed)*nfreq*2;                            "
"        size_t nvoff = numvis*nfreq*2;                                  "
"        for(size_t k=0; k<(size_t)nfw; k++) {                            "
"            size_t kfreq = kfreqoff+2*k;                                "
"            C[nvoff+kfreq] += A[ioff+kfreq]*A[joff+kfreq]-A[ioff+kfreq+1]*A[joff+kfreq+1]; "
"            C[nvoff+kfreq+1] += A[ioff+kfreq]*A[joff+kfreq+1]+A[ioff+kfreq+1]*A[joff+kfreq]; "
"        }                                                                "
"    }                                                                    ";
sources.push_back(std::pair<const char*, size_t>(kernel_code.c_str(),kernel_code.length()));
```

*Allocation sur GPU*

```
// create array buffers on the device
size_t SZA = NFEED*NFREQ*sizeof(float)*2;
cl::Buffer buffer_A(context,CL_MEM_READ_ONLY,SZA);
size_t SZC = NVIS*NFREQ*sizeof(float)*2;
cl::Buffer buffer_C(context,CL_MEM_READ_WRITE,SZC);
```

```
cl::Program program(context,sources);
std::vector< cl::Device > v_devices;
v_devices.push_back( default_device );
if(program.build(v_devices)!=CL_SUCCESS){
    std::cout<<"test_correl/ Error building: " <<
    program.getBuildInfo<CL_PROGRAM_BUILD_LOG>(default_device)<<"\n";
    throw PException("test_correl/ Error program.build "); }
}
```

*Mémoire pour les coefficients de Fourier -  
tableau A = une zone de MemMgr*

*Mémoire pour les coefficients de Fourier -  
tableau B = Matrice de visibilité*

*Compilation du kernel*



```

size_t WKGSZ=NFREQ;
while (WKGSZ>maxwksz0) WKGSZ /= 2;
if (WKGRPSZ>0) WKGSZ=WKGRPSZ;
if (NFREQ%WKGSZ != 0) {
    cout << "test_correl/Info: NFREQ not multiple of WKGSZ="<<WKGSZ;
    throw PException("test_correl: wksz NOT multiple of NFREQ");
}
size_t NFWKIT = NFREQ/WKGSZ;
size_t GLSZ = NVIS*WKGSZ;

cout << "[2.b] kernel and NDRange parameters \n ...NFreq="<<NFREQ<<" NFWKIT=NFreq/WorkItem="<<NFWKIT
    <<" NFreq*NVIS="<<NFREQ*NVIS<<" GLSZ=NVIS*WKGSZ="<<GLSZ<<endl;

cl::Kernel kernel_correl=cl::Kernel(program,"rz_correl");
kernel_correl.setArg(0,buffer_A);
kernel_correl.setArg(1,buffer_C);
kernel_correl.setArg(2,(unsigned long)NFREQ);
kernel_correl.setArg(3,(unsigned long)NFWKIT);
kernel_correl.setArg(4,(unsigned long)GLSZ);
kernel_correl.setArg(5,(unsigned long)WKGSZ);

//create queue to which we will push commands for the device.
cl::CommandQueue queue(context,default_device);

```

## *Paramètres de regroupement - niveau de parallélisme*

*On fixe les paramètres pour le kernel*

```

for (size_t l=0; l<NLOOP; l++) {    visi_gpu = zz;
    queue.enqueueWriteBuffer(buffer_C,CL_FALSE,0,SZC,visi_gpu.Data());
    for (size_t ll=0; ll<NBPAQ; ll++) {
        //DBG cout << " *DBG*  ll="<<ll<<endl;
        TArray< DTC > & sig = vsig[ll];
        queue.enqueueWriteBuffer(buffer_A,CL_FALSE,0,SZA,sig.Data());
        //DBG queue.finish(); cout << " .... done enqueueWriteBuffer(A)"<<endl;
        cl_int rcq=queue.enqueueNDRangeKernel(kernel_correl,cl::NullRange,cl::NDRange(GLSZ),cl::NDRange(WKGSZ));
        if (rcq != CL_SUCCESS) {
            if (ll<5) cout<<"test_correl/ Error enqueueNDRangeKernel (kernel_correl) : Rc="<<rcq<<" ->
"<<getCLErrorString(rcq)<<endl;        }
    }
    queue.finish();
    queue.enqueueReadBuffer(buffer_C,CL_TRUE,0,SZC,visi_gpu.Data());
}

```

*Recopie -> GPU matriceVisi (mise à zéro)*

*Recopie -> GPU des paquets, une zone = tableau 2D*

*Execution du noyau, avec parallélisme WKGSZ*



---

# Récapitulatif

---

- ❖ Le kernel et code openCL de [tstopencl.cc](http://tstopencl.cc) peut assez facilement être incorporé ds TAcq (calcul des  $N^2/2$  corrélations au lieu de  $N^2$ , prise en charge des offsets avec les BRPaquets ...)
- ❖ Résultats détaillés et quelques docs ds SophyaProgs/Eval/DivTst/README
- ❖ On tourne actuellement à 4kHz à Nançay , (deux machines de calcul de visi, 1/2 bande chacune) , avec 1.7 GFLOPS (multi-thread) / machine (~6.5% de cycle utile sur ciel)
  - ❖ 1.7 GFLOPS obtenu avec 4 BRVisiCalculator, ~420 MFLOPS / BRVisiCalculator , 2 threads en // dans chaque BRVisiCalculator
  - ❖ Sur machine bao5,6 [ 2 x Xeon E5620 4 cores @ 2.4 GHz ] , → 8 cores au total , année ~ 2012  
<https://ark.intel.com/content/www/us/en/ark/products/47925/intel-xeon-processor-e5620-12m-cache-2-40-ghz-5-86-gt-s-intel-qp.html>
- ❖ Il faut donc 10 GFLOPS pour passer à ~ 32% de cycle utile sur ciel
- ❖ On obtient ~ 11.5 GFLOPS sur Nvidia V100, 3.5 GF sur K100 (linux CC, openCL CUDA) et 4.6 GF sur AMD Radeon 575 (mac) - sur Linux, gain de 10 à plus de 20 par rapport à CPU mono-thread
- ❖ Tests à faire avec CUDA ? Optimiser le code OpenCL ? - **en cours (Automne 2020, Olivier )**
- ❖ Test à faire sur le serveur et GPU acheté pour PAON4 - **en cours (Automne 2020, Olivier)**
  - ❖ nouvelle bao2 [ 4 x Xeon Silver 4210 , 10 cores @ 2.2 GHz ] → 40 cores au total , année 2019