PAON4/IDROGEN

Corrélateur PAON4

TACQ / GPU

R. Ansari

Juillet 2020

Rappels

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

Acquisition/visibility computation (**mfacq**)

ZThread

T1 -ReadEternet

BRVisibilityCalculator

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

RAcqMemZoneMgr

T3 - Monitoring

Threads : EthernetReader, VisiCalc, Monitoring …
Task: PCIExpress ➠ Ethernet

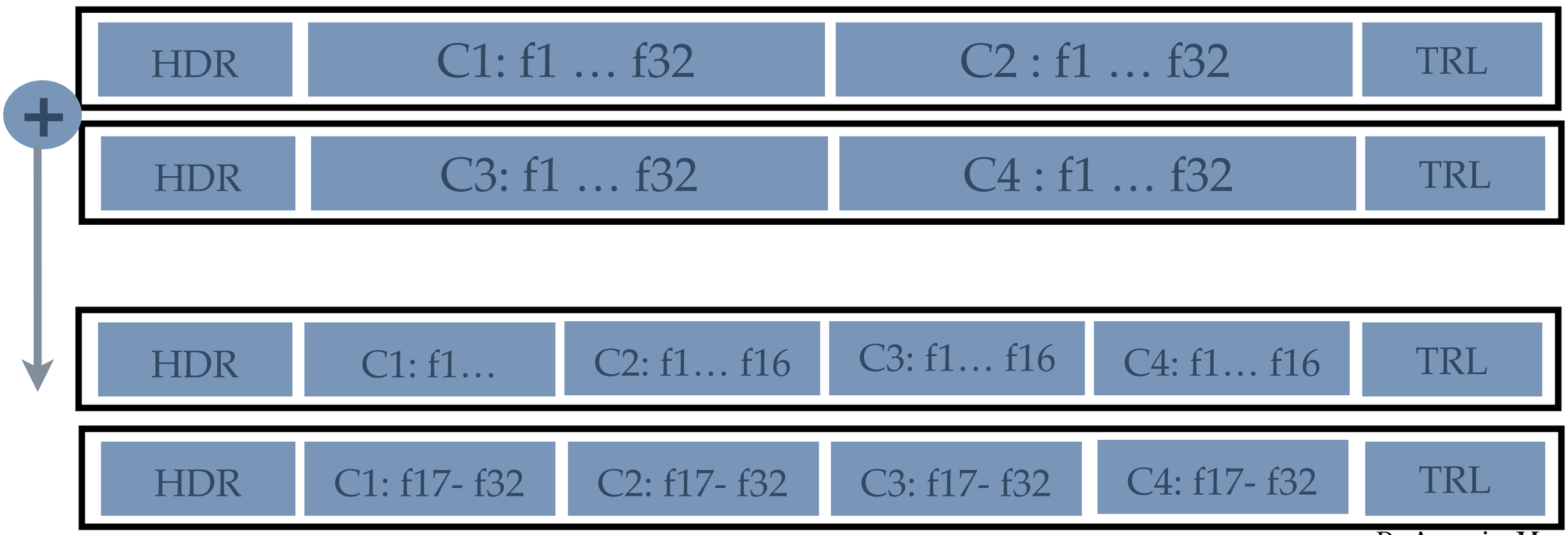
PCIExpress ↔ **DMA-Task** ↔ **Ethernet**

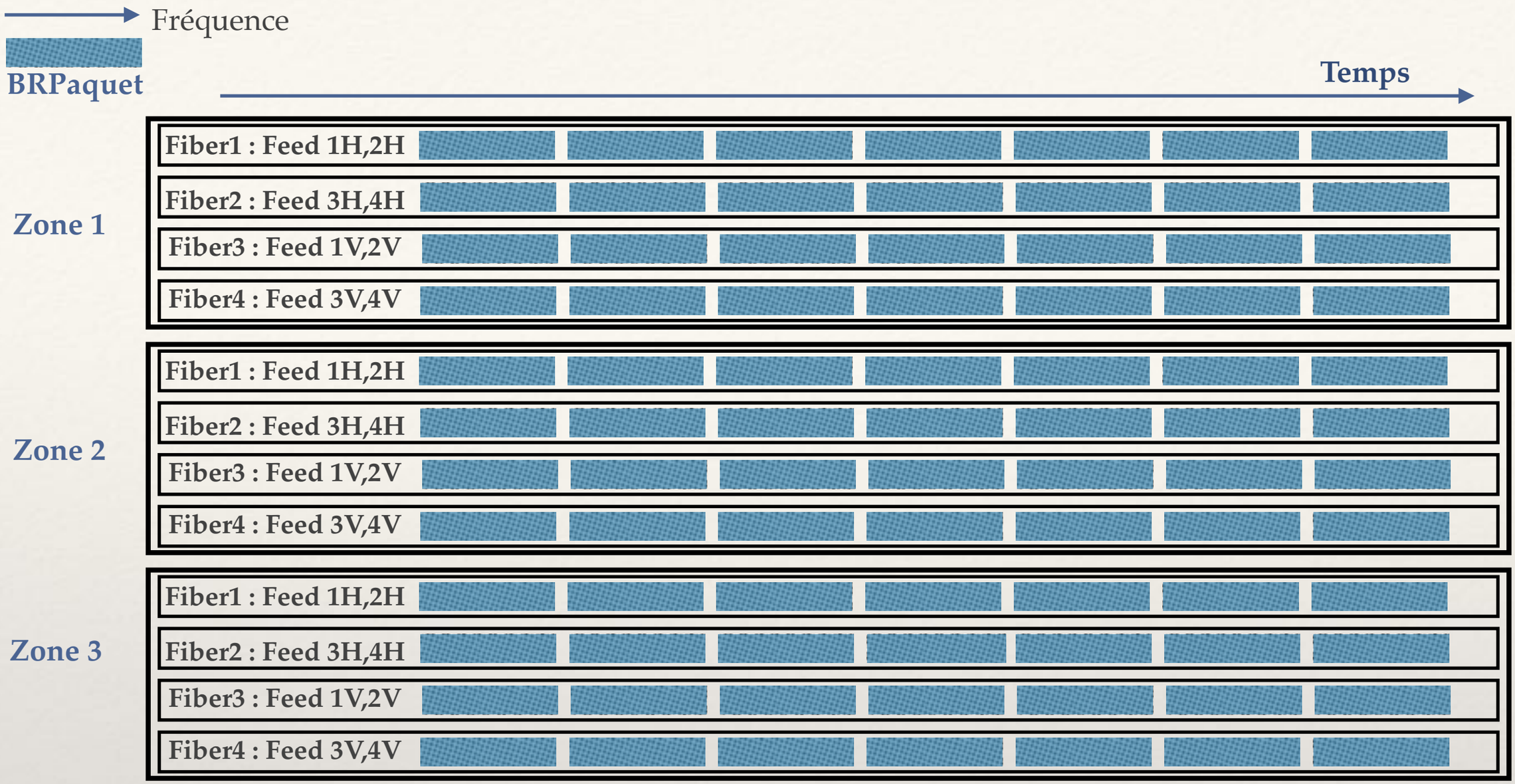
BRPaquet

- 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

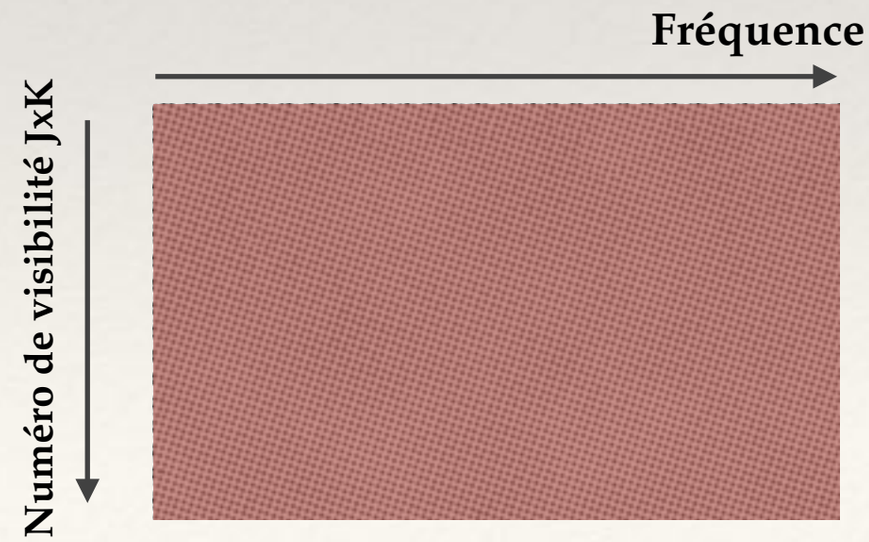




…

RAcqMemZoneMgr

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



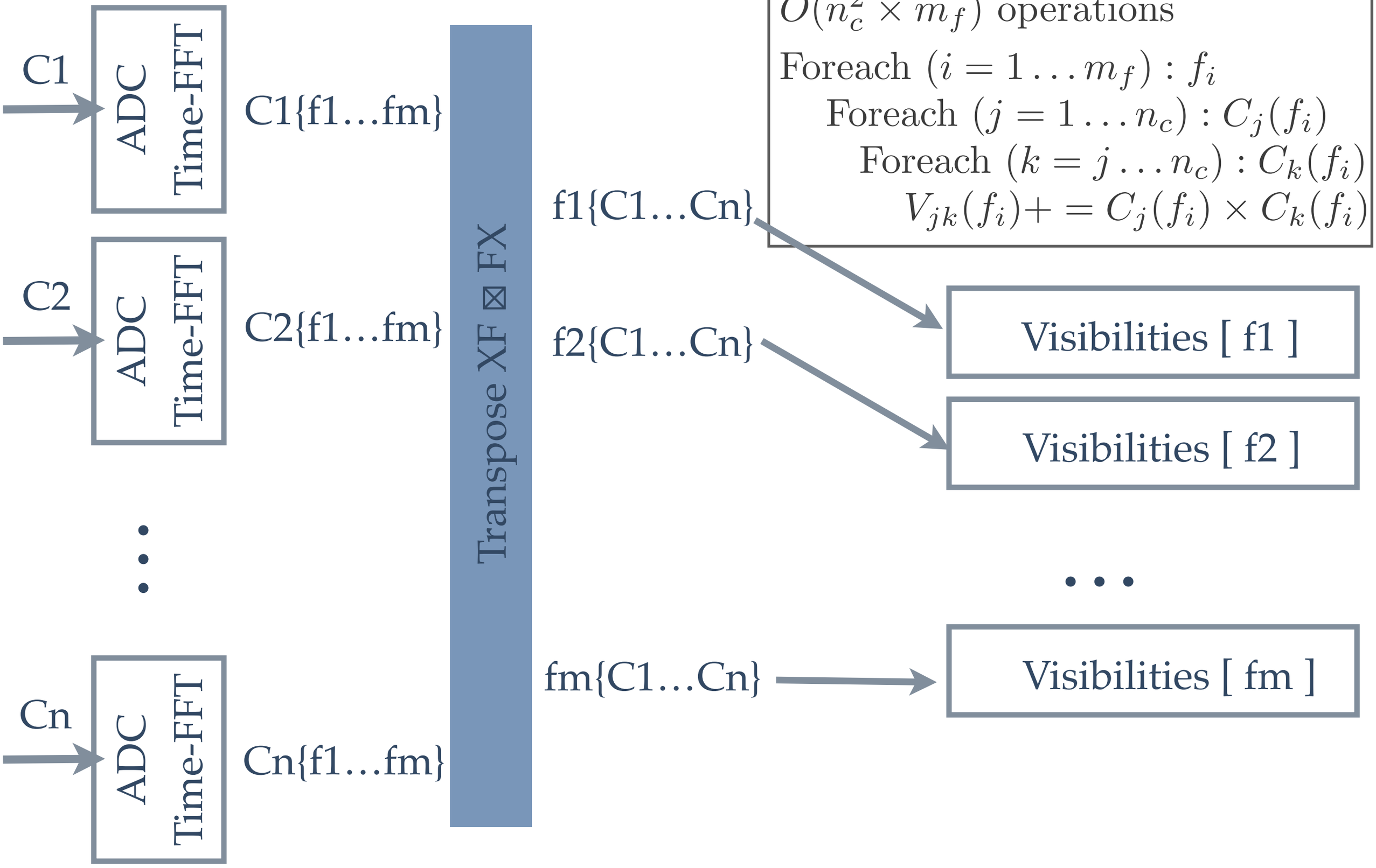
Calcul des corrélations

Classes BRVisibilityCalculator et BRVisCalcGroup

méthode BRVisibilityCalculator::execute(int tid),

~120 lignes sur les 1200 lignes de code de 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


C1
ADC
Time-FFT
C1{f1…fm}
C2
ADC
Time-FFT
C2{f1…fm}
Cn
ADC
Time-FFT
Cn{f1…fm}
Transpose XF ⊠ FX
f1{C1…Cn}
f2{C1…Cn}
fm{C1…Cn}
Visibilities:
$O(n_c^2 \times m_f)$ operations
Foreach $(i = 1 \ldots m_f) : f_i$
Foreach $(j = 1 \ldots n_c) : C_j(f_i)$
Foreach $(k = j \ldots n_c) : C_k(f_i)$
$V_{jk}(f_i) += C_j(f_i) \times C_k(f_i)$
Visibilities [f1]
Visibilities [f2]
Visibilities [fm]

Programme de test GPU/OpenCL

`SophyaProgs/Eval/DivTst/tstopencl.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);                                        "
   "     size_t kfreqoff =  get_local_id(0)*(size_t)nfw*2;                       "
   "     /* 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()));
```

GroupID = num de visibilité
LocalId = numéro de paquet

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);
```

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

Mémoire pour les coefficients de Fourier - tableau B = Matrice de visibilités

```
  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 "); }
```

Compilation du kernel

Paramètres de regroupement - niveau de parallélisme

```
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: wkgsz 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)NFEED);
 kernel_correl.setArg(3,(unsigned long)NFREQ);
 kernel_correl.setArg(4,(unsigned long)NFWKIT);
 kernel_correl.setArg(5,(unsigned long)WKGSZ);

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

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 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)
- 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 ?
- Test à faire sur le serveur et GPU acheté pour PAON4