d´OpenCL à SYCL

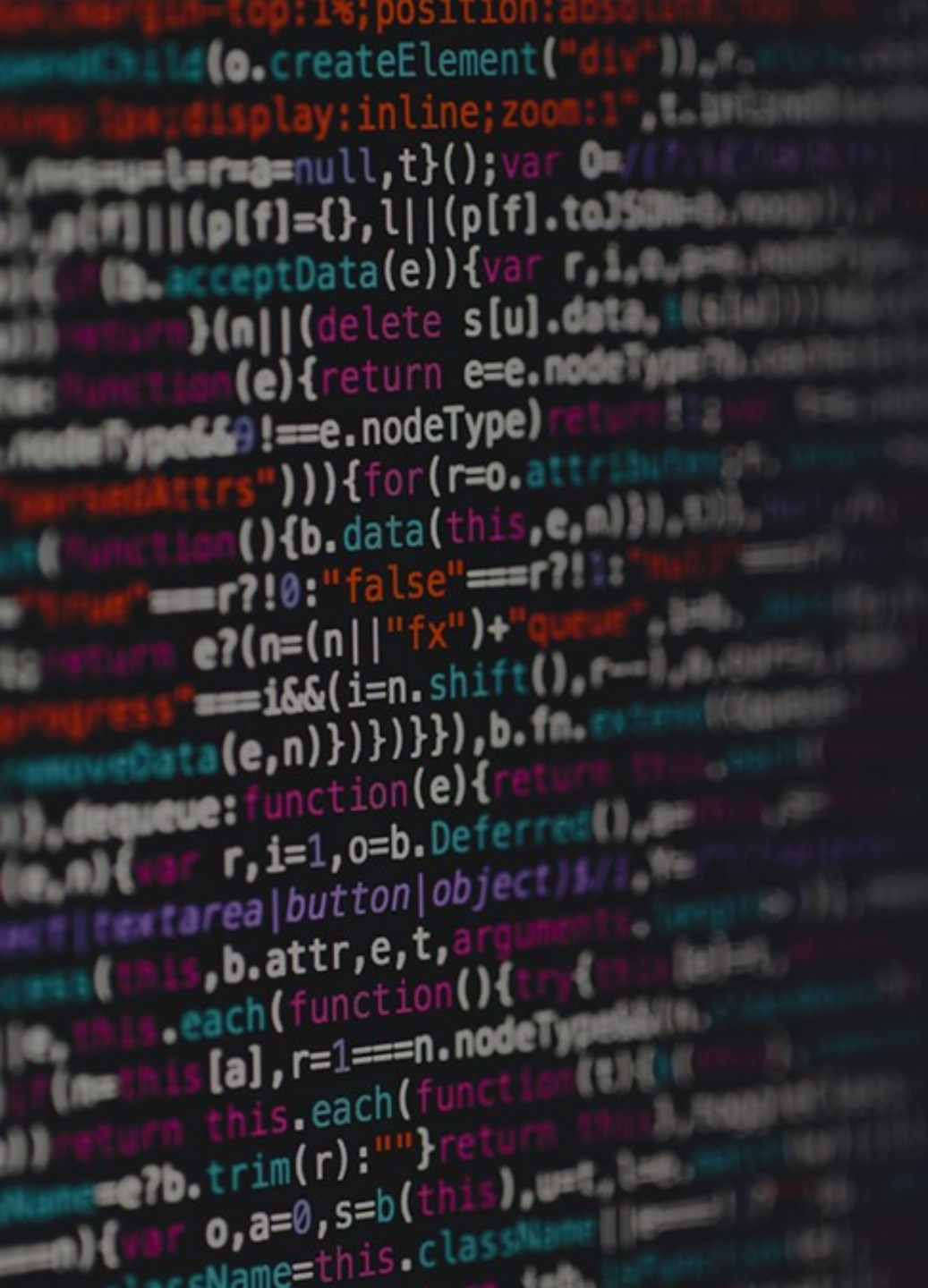
Programmation portable des accélérateurs de calcul, pour le calcul intensif (hpc) et haut-débit (htc)

David Chamont, IJCLab, Reprises

13ème Journées Informatique IN2P3/IRFU

Plan

- Pourquoi les accélérateurs ? OpenCL ? SYCL ?
- Hello World avec OpenCL & SYCL
- Focus sur les échanges de données



Pourquoi ?

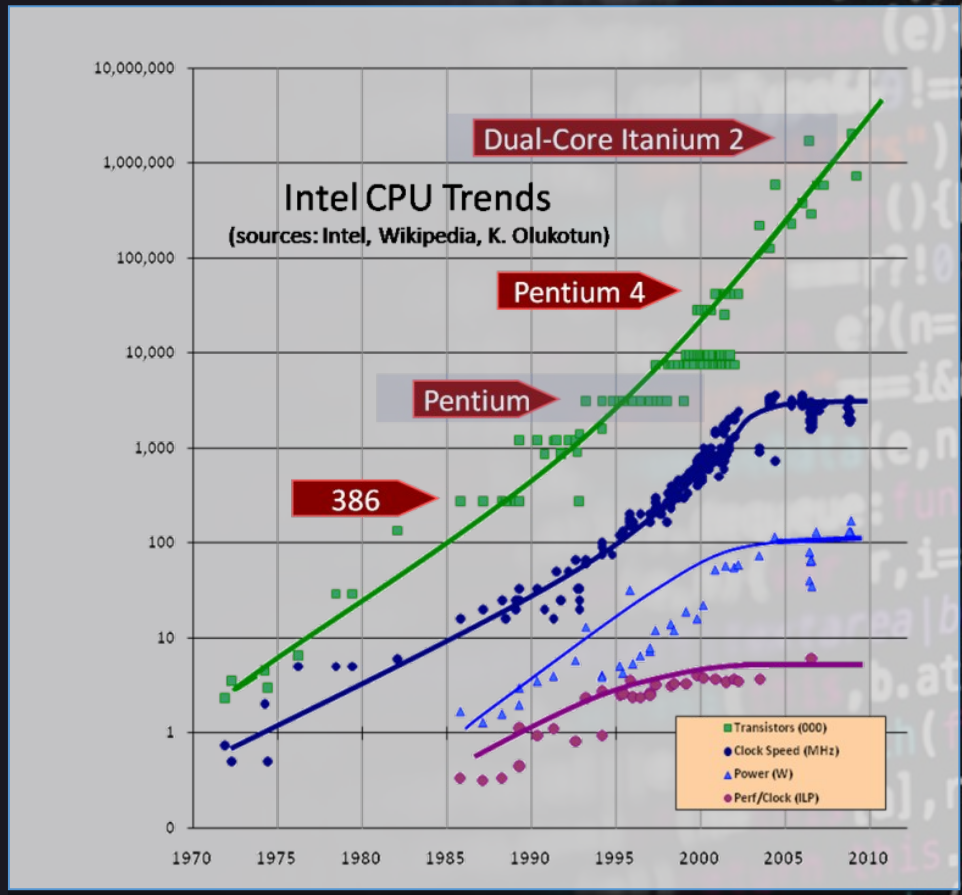
Accélérateurs, OpenCL, SYCL

David Chamont, IJCLab, Reprises
13ème Journées Informatique IN2P3/IRFU

The free lunch is over

Herb Sutter, 2005

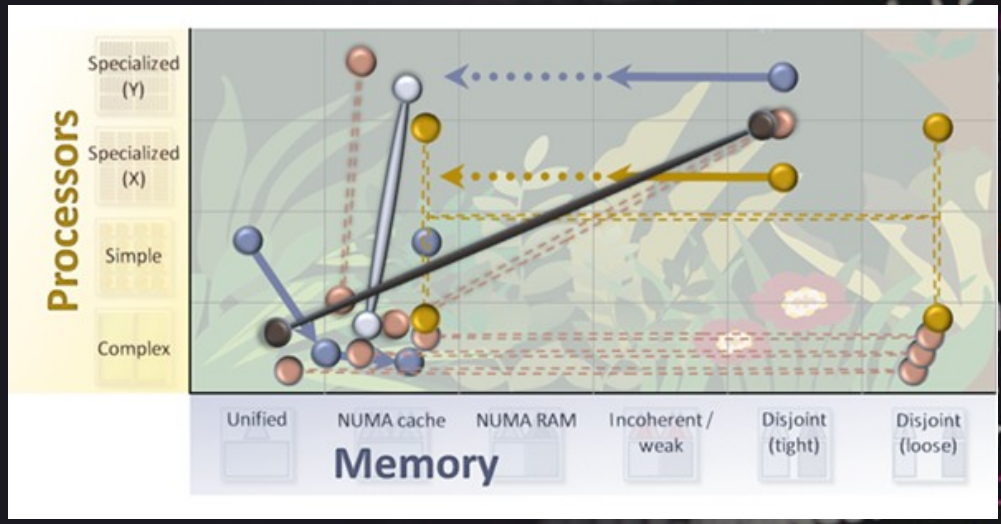
- Matériel : toujours plus de coeurs
  - La fréquence d’horloge se stabilise
  - Le nombre de transistors continue de croitre, permettant de multiplier les coeurs.
  - La mémoire vive ne suit pas...
- Logiciel : parallélisme obligé
  - Nos applications ne bénéficent plus automatiquement des progrès du matériel.
  - Retour en force de la programmation concurrente.



Welcome to the jungle

Herb Sutter, 2013

- Matériel : toujours plus complexe
  - Mélange de petits et gros coeurs, addition d'accélérateurs, avec canaux de communication dédiés.
  - La mémoire ne peut plus être considérée comme une boîte noire homogène.
  - La précision ne peut plus être "double" partout et tout le temps.



- Logiciel : forcémment hybride
  - Les accélérateurs ne peuvent pas tout. Il faut les combiner avec le multi-threading, la vectorisation, le calcul intensif, distribué...

Jungle matérielle

- Intel rachète Altera, NVidia rachète ARM...
- Google/Amazon/Qualcomm inventent leurs propres puces...
- Course à la puissance des accélérateurs dans les supercalculteurs :
  - Nvidia A100 : 19,5 Tflops.
  - AMD MI250X : 47,9 Tflops.
  - Intel Ponte Vecchio : 45 Tflops.
  - Réplique NVidia à venir au printemps...

Jungle logicielle
(pour les accélérateurs)

- Applications déjà instrumentées : Matlab, Mathematica, ...
- Directives : OpenACC/OpenMP4.
- Bibliotheques C++ : CuBLAS..., ArrayFire, TensorFlow, Kokkos, Raja, ...
- Extensions C++ : SyCL, Intel DPC++...
- Programmation explicite de bas niveau : OpenCL, Cuda, Vulkan.

Reprises 

Projet et groupe IN2P3, qui essaie de se frayer une piste dans la jungle, à la recherche du bon équilibre entre Performance, Portabilité, Pérénité, Productivité, Précision…

- Depuis 4 ans
- Une douzaine d’ingénieurs (IJCLAB, IPHC, LAPP, LPC, LLR, LUPM, SUBATECH, *LPNHE*, *L2IT*)

Commentaires généraux

- Les accélérateurs offrent le meilleur rapport Flops/Watt... si ils tournent au maximum de leurs capacités, ce qui exige entre autres :
  - Transfert des données d'entrées (ce qui a un coût),
  - Suffisamment de calcul (haute intensité arithmétique),
  - Gérer les coeurs CPU qui se disputent l'accès au GPU,
  - Les occuper pendant qu'ils attendent les résultats.
- Cette programmation peut sembler fastidieuse, mais l'effort paiera, même si le code reste finalement sur CPU.

“Hello World”

Elévation au carré des éléments d´une collection

David Chamont, IJCLab, Reprises
13ème Journées Informatique IN2P3/IRFU

OpenCL 1/4

```
#include <CL/opencl.h>

const char * KernelSource = "                    \\n" \\
"   __kernel void square(                        \\n" \\
"     __global float* input,                     \\n" \\
"     __global float* output,                    \\n" \\
"     const unsigned int count)                  \\n" \\
"   {                                            \\n" \\
"     int i = get_global_id(0);                  \\n" \\
"     if(i < count)                              \\n" \\
"       output[i] = input[i]*input[i];            \\n" \\
"   }                                            \\n" \\
"                                                \\n";

...
```

OpenCL main 2/4

```
.....

int main(int argc, char** argv)
{
  // Prepare host data
  float data[1024], results[1024];
  .....

  // Prepare kernel
  int err;
  cl_device_id device_id;
  err = clGetDeviceIDs(NULL,CL_DEVICE_TYPE_GPU,1,&device_id,NULL);
  cl_context context = clCreateContext(0,1,&device_id,NULL,NULL,&err);
  cl_command_queue queue = clCreateCommandQueue(context,device_id,0,&err);
  cl_program prog = clCreateProgramWithSource(context,1,(const char **)&KernelSource,...
  err = clBuildProgram(prog,0,NULL,NULL,NULL,NULL);
  cl_kernel kernel = clCreateKernel(prog,"square",&err);

  // Device io arrays
  cl_mem input = clCreateBuffer(context,CL_MEM_READ_ONLY,sizeof(float)*count,...
  cl_mem output = clCreateBuffer(context,CL_MEM_WRITE_ONLY,sizeof(float)*count,...
  .....
```

OpenCL main 3/4

```
.....

 // Write our data set into the input array in device memory
 err = clEnqueueWriteBuffer(queue,input,CL_TRUE,0,sizeof(float)*count,data,0,...

 // Set arguments to kernel
 err  = clSetKernelArg(kernel,0,sizeof(cl_mem),&input);
 err |= clSetKernelArg(kernel,1,sizeof(cl_mem),&output);
 err |= clSetKernelArg(kernel,2,sizeof(unsigned int),&count);

 // Queue kernel and wait for its execution end
 size_t global = count;
 err = clEnqueueNDRangeKernel(queue,kernel,1,NULL,&global,NULL,0,NULL,NULL);
 clFinish(commands);

 // Read back the results from the device to verify the output
 err = clEnqueueReadBuffer(queue,output,CL_TRUE,0,sizeof(float)*count,results,0,...

 .....
```

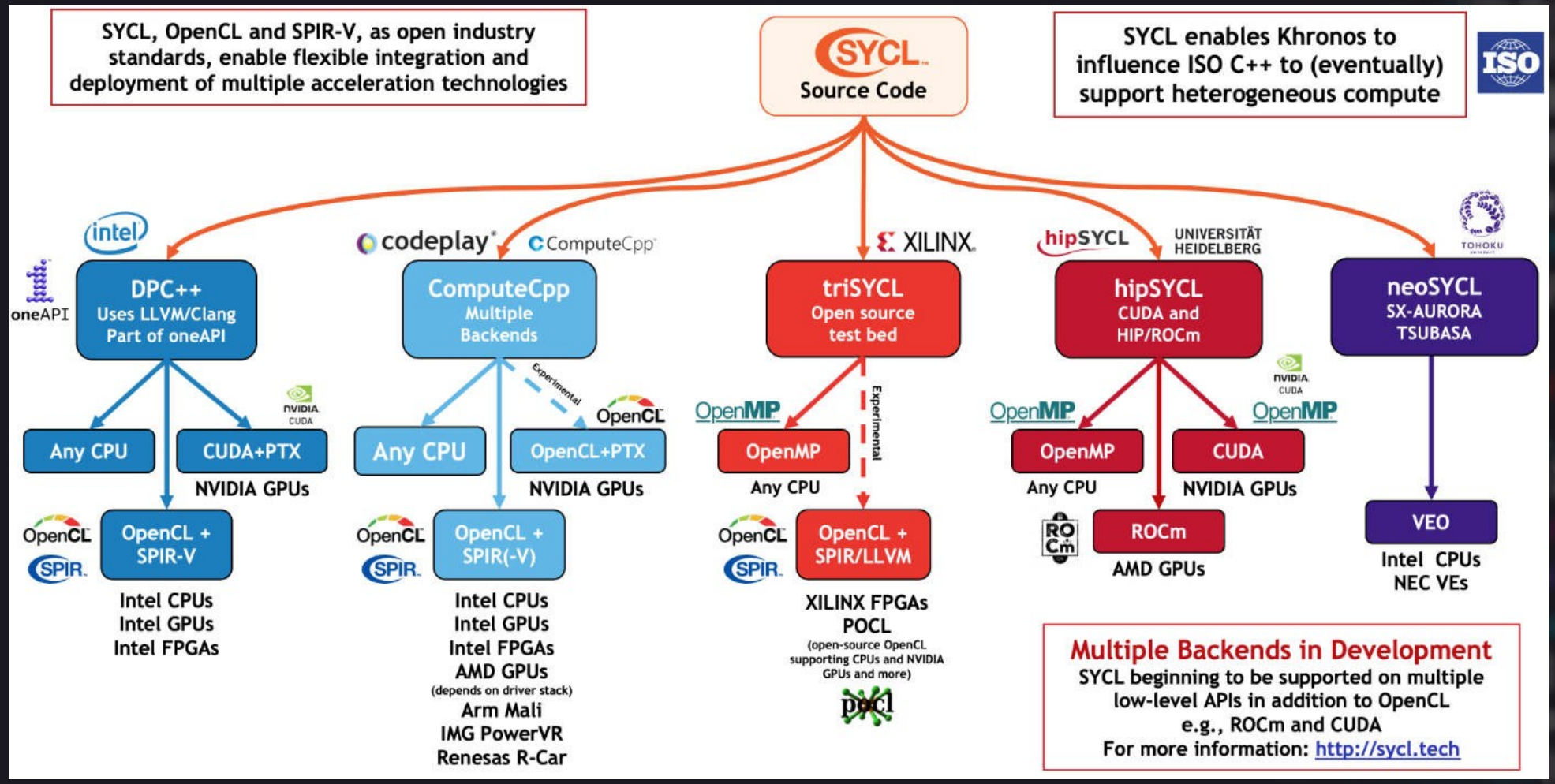
OpenCL main 4/4

```
 .....

  // Process results
  .....

  // Shutdown and cleanup
  clReleaseMemObject(input);
  clReleaseMemObject(output);
  clReleaseProgram(prog);
  clReleaseKernel(kernel);
  clReleaseCommandQueue(commands);
  clReleaseContext(context);
  return 0 ;
}
```

Commentaires OpenCL

- Le C est portable mais verbeux.
- Aucun physicien n’écrira jamais de l’OpenCL ! (ni aucun ingénieur ?)
- Nvidia ne s’empresse pas de fournir des pilotes OpenCL dernier cri pour ses cartes.

Implémentations SYCL



SYCL 1/2

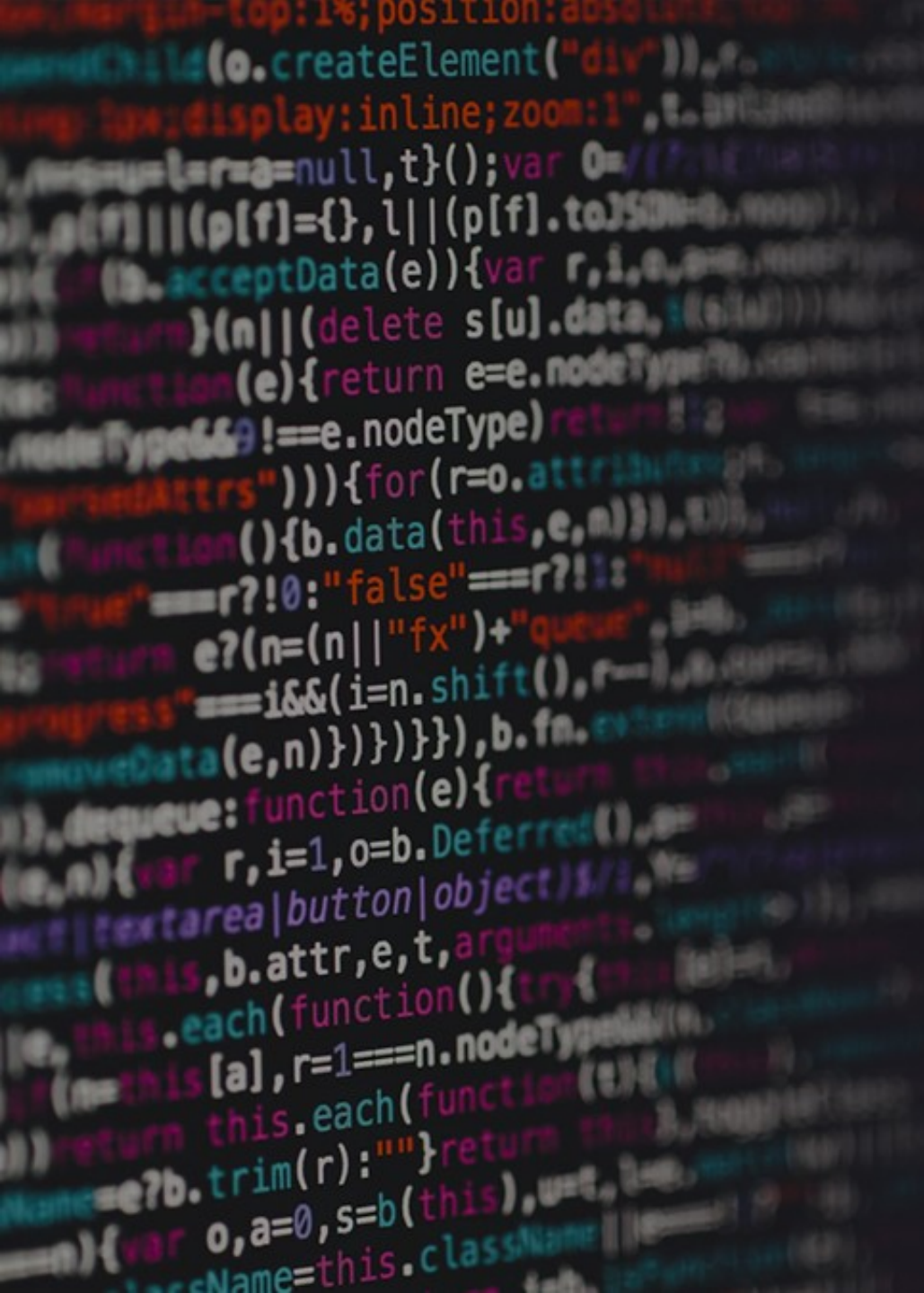
```
#include <sycl.hpp>

using namespace cl::sycl;

void traitement( auto const & data, auto & results ) {
  buffer d_data(data), d_results(results) ;

  queue myQueue ;
  myQueue.submit( [&]( handler & h ) {

    // data accessors
    accessor a_data(b_data, h, read_only);
    accessor a_results(b_results, h, write_only, no_init);

    // kernel
    h.parallel_for( range<1>(1024), [=]( auto i ) {
      uint index = i[0];
      a_results[index] = a_data[index] * a_data[index];
    });

  });
}
```

SYCL 2/2

```
.....

int main() {

  // prepare host data
  std::vector<double> data(1024) ;
  .....
  std::vector<double> results(1024) ;


  // Device buffers
  traitement(data,results) ;


  // process results
  .....
}
```

Commentaires SYCL

- Nécessite la connaissance de C++17, et des compilateurs spéciaux.
- La simplification de l´écriture de l'application se paie par une installation complexe des multiples implémentations et backends.
- La plupart des applications modernes sont limitées par les échanges de données. C'est un aspect essentiel dans l'utilisation d'accélérateurs additionnels.



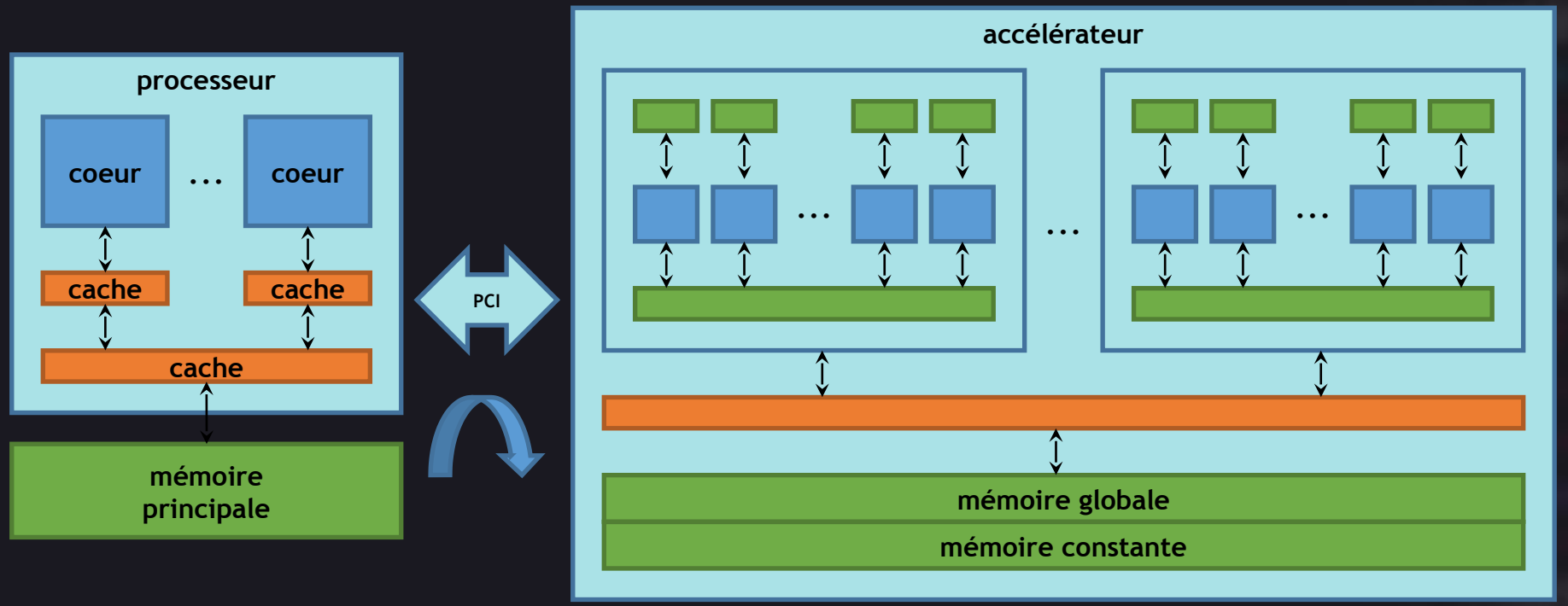
Echanges de données

(crédits : Sylvain Joube)

David Chamont, IJCLab, Reprises

13ème Journées Informatique IN2P3/IRFU

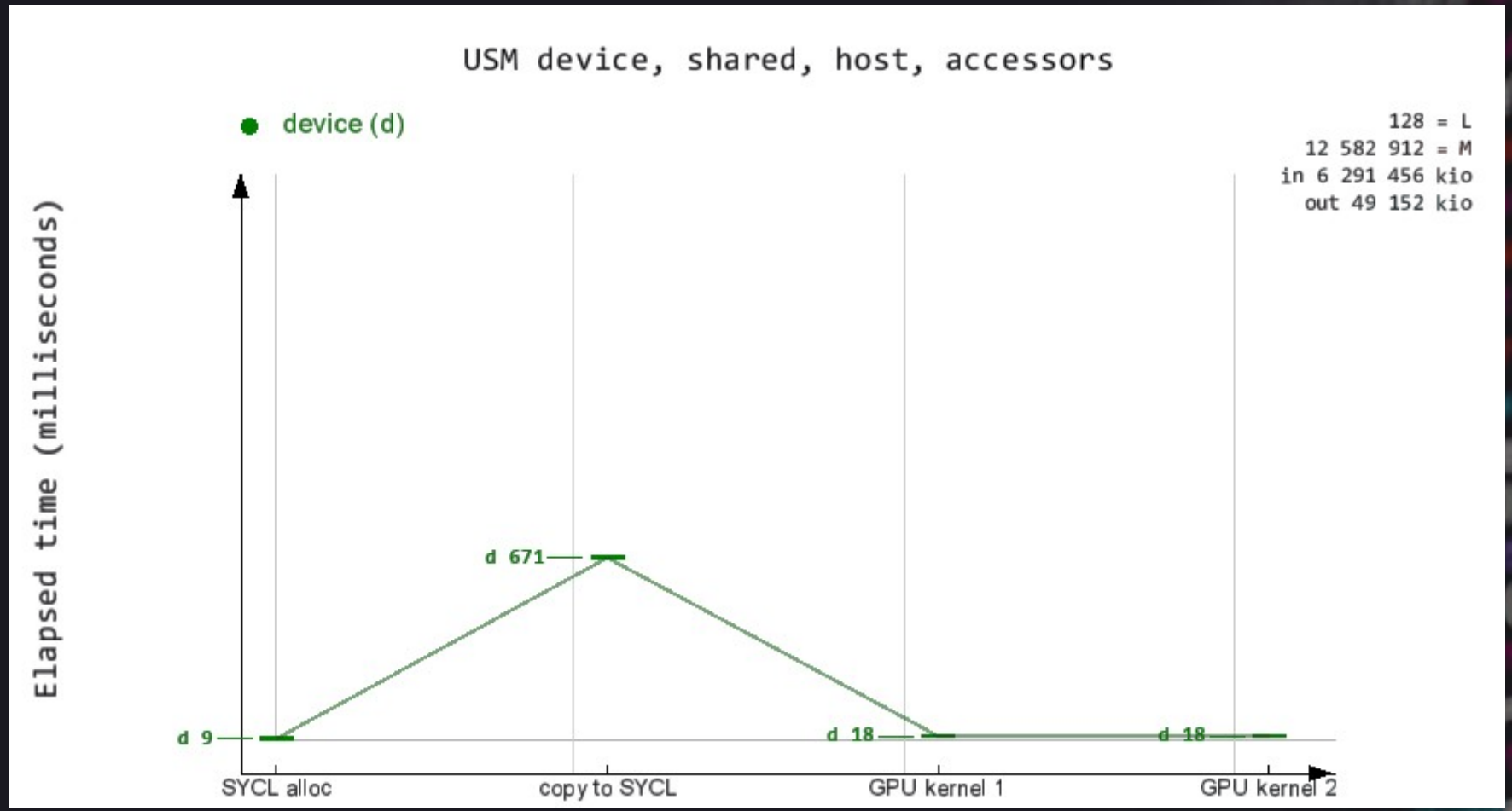
Accérateurs additionnels



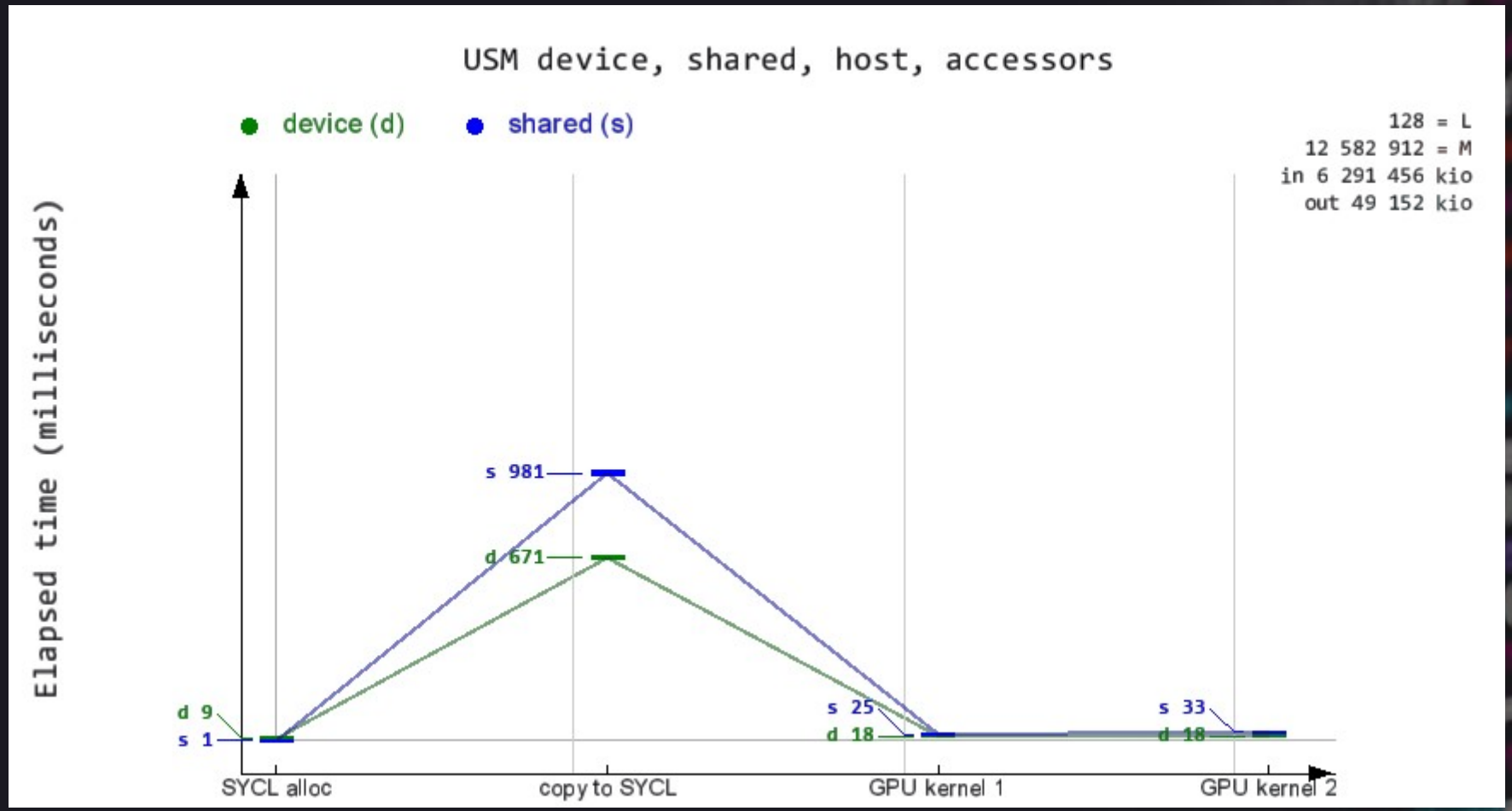
Modèles mémoire SYCL

- Buffers & Accessors
  automatise les échanges, mais exige une refonte des structures
  attendu : pour un nouveau code, ou si on préfère les flux de données
- Unified Shared Memory (2020)
  - USM Shared
    synchronise les données entre hôte et accélérateur, telles qu'elles sont
    attendu : pour un code existant, et/ou si on prefere un graphe de tâches
  - USM Device
    laisse tout le contrôle au programmeur
    attendu : plus de travail, plus de performance
  - USM Host
    rend les données de l´hôte accessible à l'accélérateur
    attendu : pour quelques cas particuliers

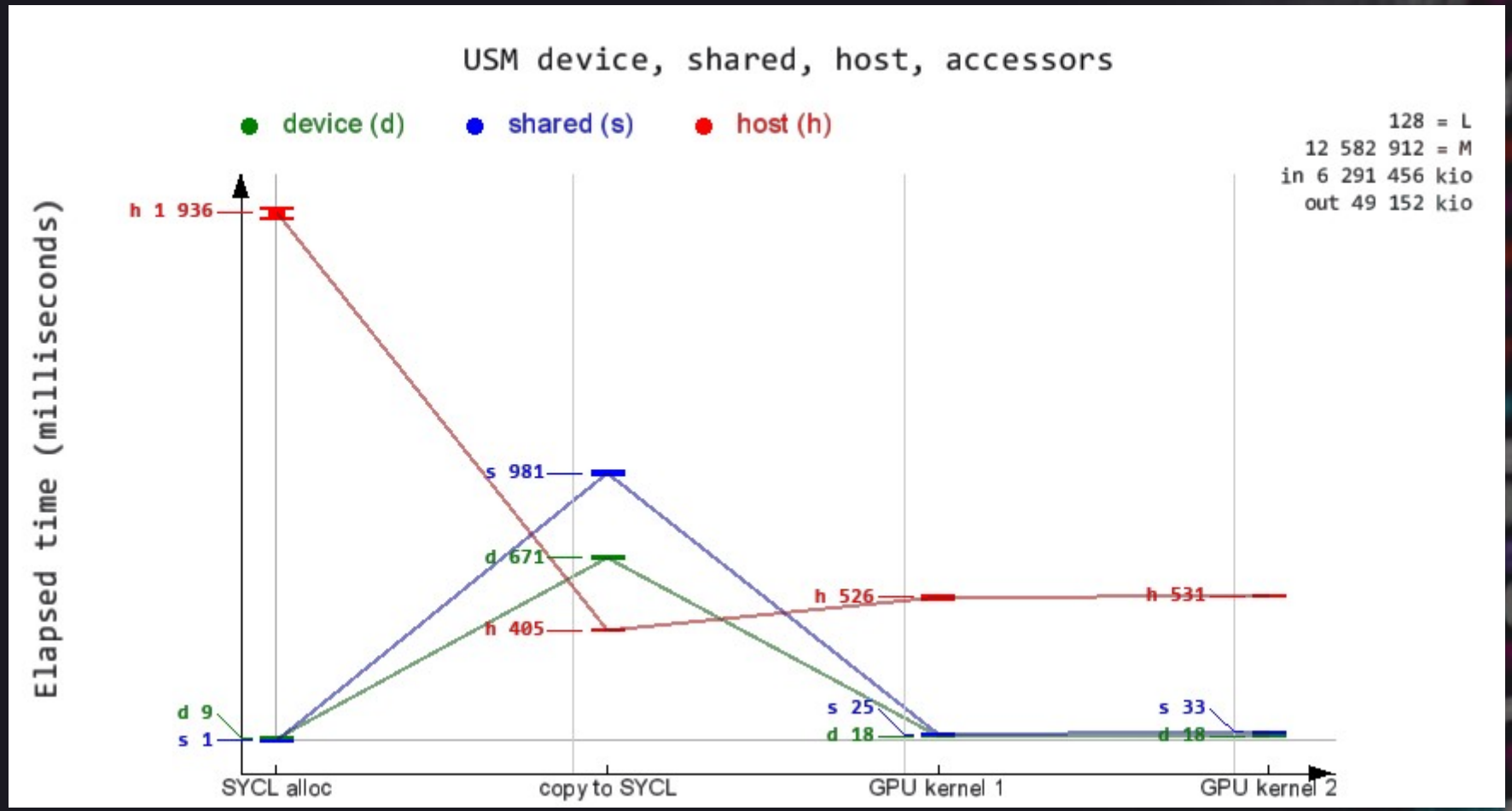
USM Device


USM device, shared, host, accessors
device (d)
128 = L
12 582 912 = M
in 6 291 456 kio
out 49 152 kio
Elapsed time (milliseconds)
d 671
d 9
d 18
d 18
SYCL alloc
copy to SYCL
GPU kernel 1
GPU kernel 2

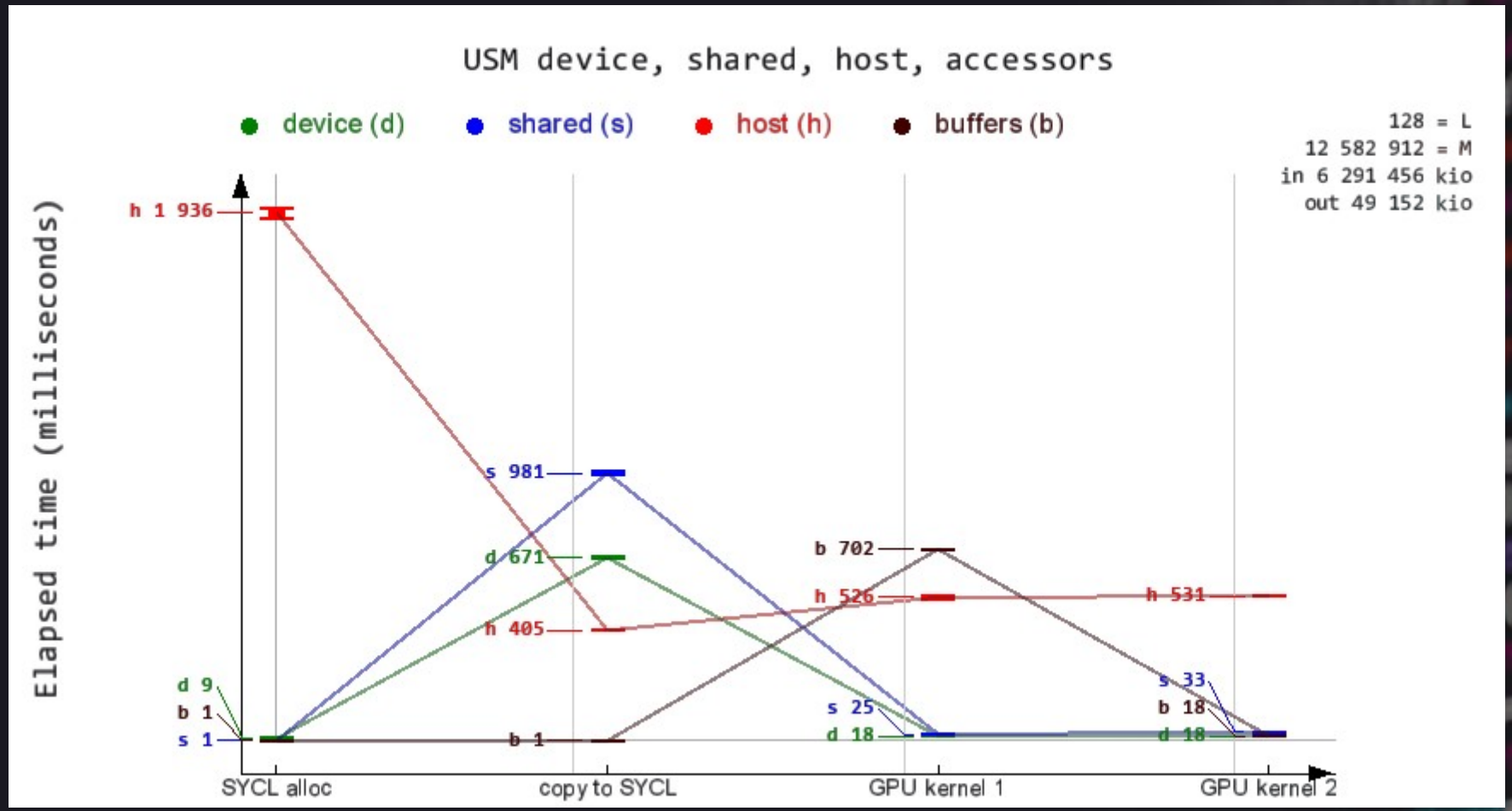
+ USM Shared


USM device, shared, host, accessors
device (d)
shared (s)
128 = L
12 582 912 = M
in 6 291 456 kio
out 49 152 kio
Elapsed time (milliseconds)
s 981
d 671
d 9
s 1
s 25
d 18
s 33
d 18
SYCL alloc
copy to SYCL
GPU kernel 1
GPU kernel 2

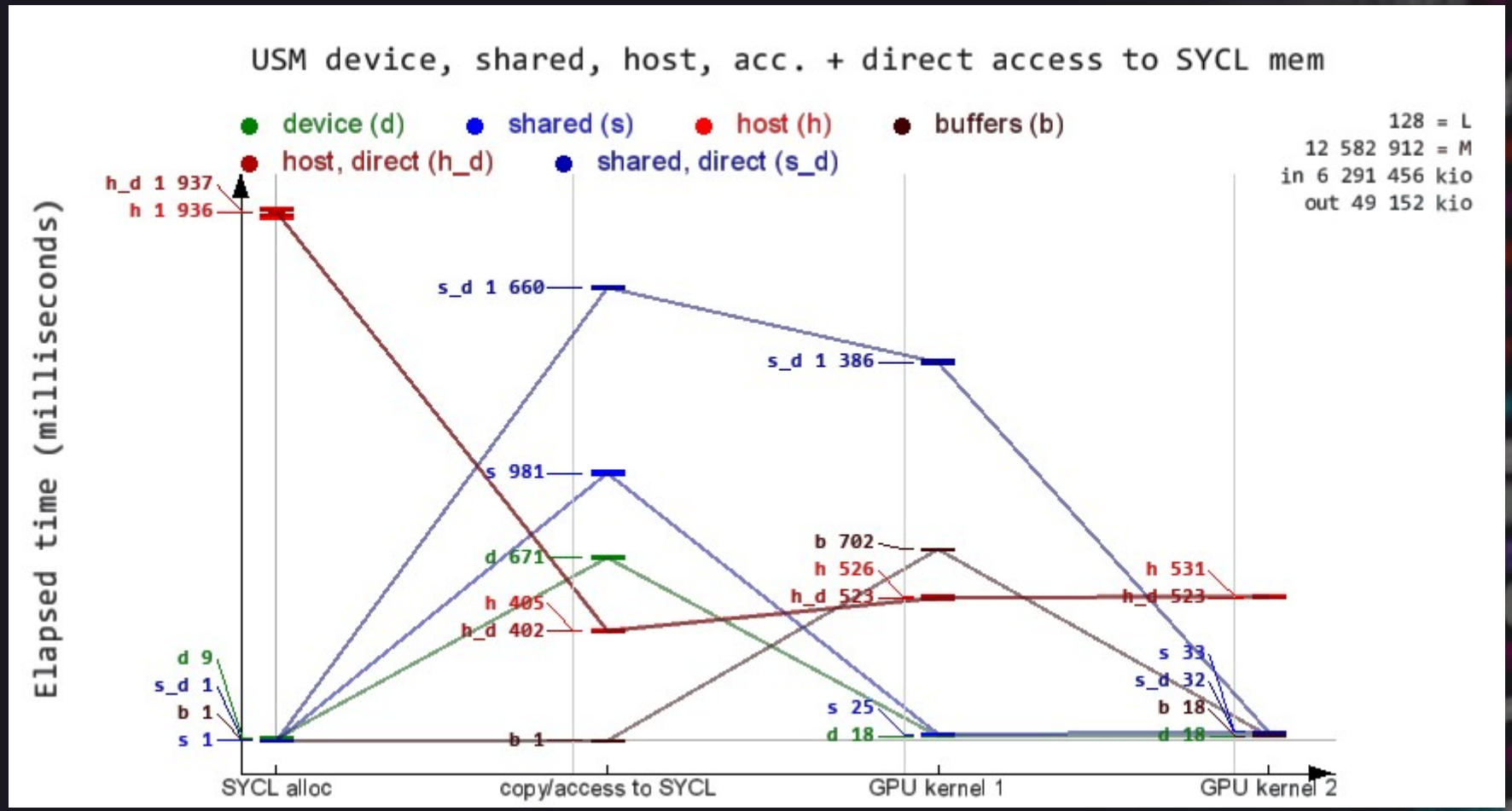
+ USM Host


USM device, shared, host, accessors
device (d)
shared (s)
host (h)
128 = L
12 582 912 = M
in 6 291 456 kio
out 49 152 kio
Elapsed time (milliseconds)
h 1 936
s 981
d 671
h 526
h 531
h 405
d 9
s 1
s 25
d 18
s 33
d 18
SYCL alloc
copy to SYCL
GPU kernel 1
GPU kernel 2

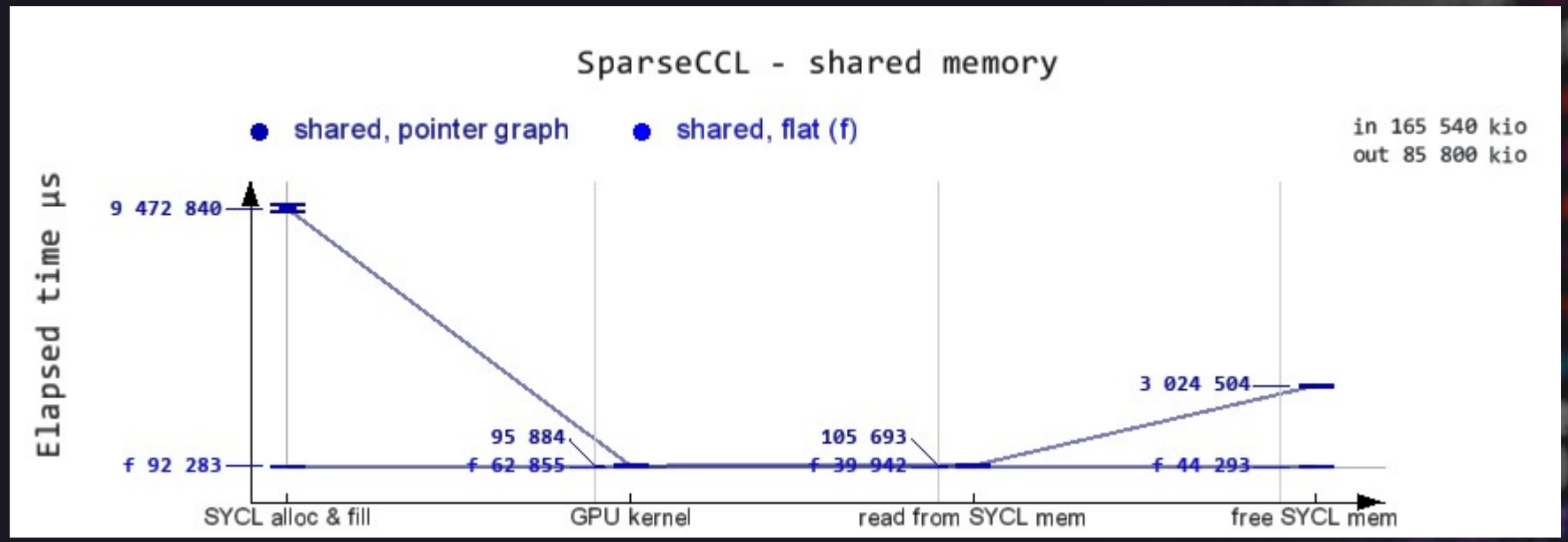
+ Buffers & Accessors



+ Host & Shared in-place



Portabilité != performance



En cours & à venir

- USM shared prefetch & mem_advice.
- USM device avec recouvrement explicite calculs/échanges.
- Diversification des implémentations SYCL testées, diversification du matériel testé.
- Elargissement des algorithmes de calcul testés (SparseCCL, Seeding, Tracking, …) et des motifs d’accès aux données.

Merci de votre attention

Des questions ?

David Chamont, IJCLab, Reprises
13ème Journées Informatique IN2P3/IRFU