École Informatique IN2P3 2016
OpenCL & MPI

G. Grasseau

Laboratoire Leprince-Ringuet
CNRS/IN2P3, École Polytechnique

OpenCL & MPI

- Introduction (motivations)
- Part 1: OpenCL
- Part 2: Hands-on OneDevice
- Part 3: Hands-on MultipleDevices
- Part 4: MPI & OpenCL
- Conclusion

OpenCL & MPI

Introduction

- HPC Context
- Accelerator Hardware landscape: Emerging technologies
- Accelerator Software landscape

Introduction
Objectives

- More familiar with OpenCL
- HPC Extensible application (scalable)
- Give you inputs to make choices
- Not focus on specific HW
- CUDA developers will not be disoriented

- The Hands-on application is specific to HEP
- … but the main // mechanisms are provided
- Hands-on real/substantial "template" for your applications
- Focus on OpenCL, MPI (for scalable app.) presented quickly (easier)

Introduction
HPC & many-core / GPU devices

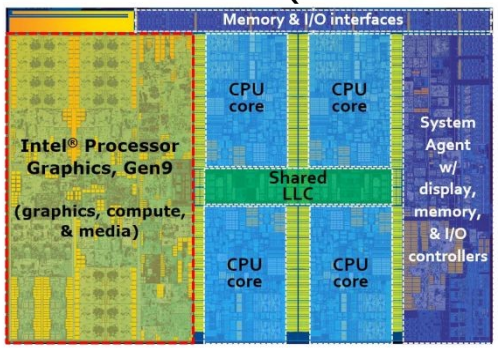
- Speaking about High Performance Computing (HPC) applications → $>10^4$ CPU hours
- High acceleration speed-up ($> 10^7$ cores) thanks to //ism → numerical scheme, refactoring
- Hybrid model (MPI+OpenMP)

- Increasing size of supercomputer HPC/HTC is not a economic model
- Power Supply > 6 MW for 2.10^5 cores
- Top 500: Tianhe-2, 18 MW
- Gflops → Gflops/Watt
- Frequency decrease since 2005 < 3 Ghz:
  - Vector operations
  - Many-(low consumption) core
- How HPC application take into account accelerators
- Accelerator device durable technology ?

Technology Evolution → strategic operation for research teams

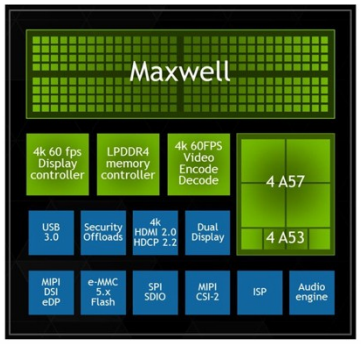
Introduction
Emerging technologies

CPU + GPU (AMD, Intel)



Integrated systems

Nvidia Tegra X1

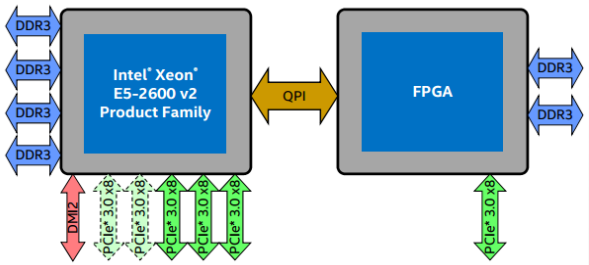


1TFlops / 10 W

Intel KNL (based on Atom)

Many-core systems

Intel Xeon + FPGA



Computing power

- Key points:
- Performance (Moore law)
- Exploitation cost: power (frequency)
- Market trends (other): games, embedded systems, smart phones, tablets, industry(Big data)

Technology Evolution : vector(GPUs), many-cores, FPGA vs Power Supply?

Introduction
Accelerator SW Ecosystem (1)

Directive approach

OpenACC
Directives for Accelerators

- Allinea, AMD, CRAY Inc, NVIDIA, ORNL, Rogue Wave, Sandia National Lab., CSCS, TOTAL
- Incremental ...
- … often lead to refactoring the code
- Restricted C for kernels
- Major drawback: **Intel** doesn't (or plan) support OpenACC (must combine with OpenMP to use cores //ism)

```
#pragma acc data copy(A) create(Anew)
while ( error > tol  && iter  <  iter_max ) {
 error = 0.0;
#pragma acc kernels
 {
# pragma acc loop
  for (  int  j = 1; j < n-1;  j++ )  {
   for (  int  i = 1; i < m-1; i++ )  {
    Anew[j][i] = 0.25*( A[j][i+1] + A[j][i-1]
                      + A[j-1][i] + A[j+1][i]);
    error = fmax( error,
                  fabs( Anew[j][i]-A [j][i] );
   }
  }
# pragma acc loop
  for ( int j = 1; j < n-1; j++) {
   for (int = i; i < m-1; i++ )  {
    A [j] [i] = Anew [j] [i];
   }
  }
 }
 iter++;
}
```

https://www.olcf.ornl.gov/wp-content/training/electronic-structure-2012/IntroOpenACC.pdf

Introduction
Accelerator SW Ecosystem (2)

Directive approach

OpenMP

- Specification OpenMP 4.0 - July 2013 (2011 → ...)
- Wide community (user & vendors) but not supported NVidia (OpenACC)
- More complex 248 p. only for OMP 4.0

```
float compute(int N, float *mat) {
  int i, j, half=N/2;
  float c, s, sum[2];
# pragma omp parallel num_threads(2) private(s)
  {
    int nthd = omp_get_thread_num();
    int start=nthd*half;
    int end=(nthd+1)*half-1;
# pragma omp target
    map(to:start,end,c,half) map(from:s)
    map(tofrom:mat[start:end])
    device(nthd)
    {
      s = 0.0;
# pragma omp parallel for
      for (int k=start;k<=end;k++) {
        mat[k] += c;
        s += mat[k];
      }
    }
    sum[nthd] = s;
  }
  return psum[0]+psum[1];
}
```

http://press3.mcs.anl.gov/computingschool/files/2014/01/OpenMP-40-features-ATPESC-final-v2.pdf

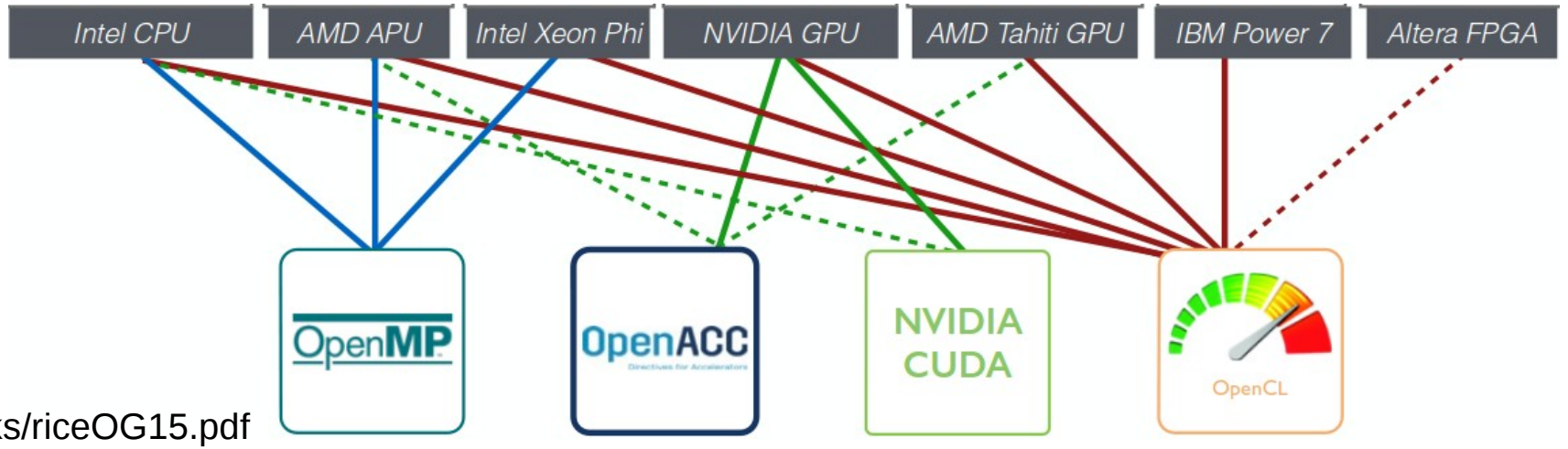
Introduction
Accelerator SW Ecosystem (3)

http://openmp.org/wp/openmp-compilers/

Vendors/Source	Compiler	Information
GNU	gcc	GCC 4.9.1, OpenMP 4.0 fully supported (but for Intel targets, for NVidia targets use OpenACC) GCC 6.1, OpenMP 4.5 will be supported (not Fortran)
IBM	XL C/C++ & Fortran	OpenMP 4.0 is partially supported on Linux (little endian)
Intel	C/C++ & Fortran	OpenMP 4.0 supported in version 16 compilers.
PGI	C/C++ & Fortran	Only OpenMP 3.1
CRAY	Cray C/C++ & Fortran	Cray Compiling Environment (CCE) 8.4 (September 2015) supports OpenMP 4.0. OpenMP is on by default.
LLVM	Clang 3.8	OpenMP 3.1 and some elements of OpenMP 4.0

Implementations exist but NOT SUPPORT BOTH main devices: Xeon Phi, NVidia GPUs, AMD GPUs, Xeon, ...

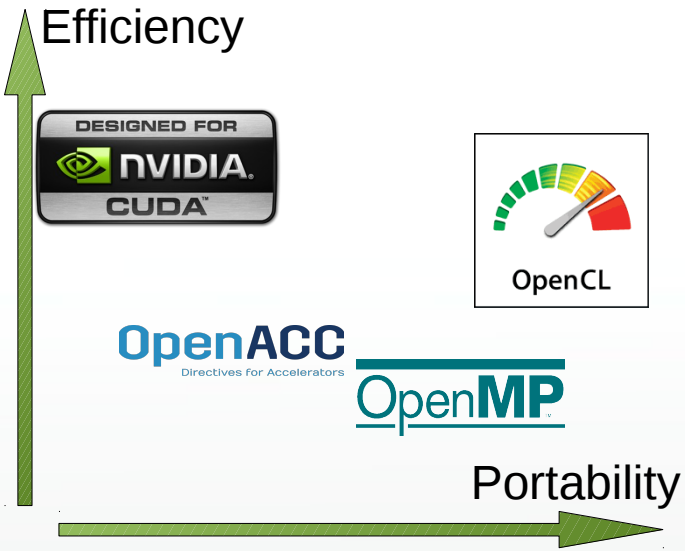
Introduction
Choosing a SW technology



http://libocca.org/talks/riceOG15.pdf

Key points

- Efficient & Portable application (changing technologies)
- Standard free (cheap) technology
- Easy development (//ism not easy task)
- Development tools (debugger, performance analysis)



Introduction
High-level portable API

API	Type	Front-ends	Kernel	Back-ends
Kokkos	ND arrays	C++	Custom	CUDA & OpenMP
VexCL	Vector class	C++	-	CUDA & OpenCL
RAJA	Library	C++	C++ Lambdas	CUDA, OpenMP, OpenACC
OCCA2	API, Source-to-source, Kernel Languages	C,C++,C#, F90, Python,MATLAB, Julia	OpenCL, CUDA,& custom unified kernel language	CUDA, OpenCL, pThreads,OpenMP, Intel COI
CU2CL *	Source-to-source	App	CUDA	OpenCL
Insieme	Source-to-source compiler	C	OpenMP,Cilk, MPI, OpenCL	OpenCL,MPI, Insieme IR runtime
Trellis	Directives	C/C++	#pragma trellis	OpenMP, OpenACC, CUDA
OmpSs	Directives + kernels	C,C++	Hybrid OpenMP, OpenCL, CUDA	OpenMP, OpenCL, CUDA
Ocelot	PTX Translator	CUDA	CUDA	OpenCL

DSL

Compiler

OpenCL & MPI

Part 1: OpenCL

- Context: history, implementations
- OpenCL concepts: Abstract model
- OpenCL API:
  host part to handle the devices
- OpenCL Kernels
- Case study: reduction

OpenCL Introduction history

OpenCL was initially developed by Apple Inc.

OpenCL 1.0 (December 2008)

- Apple , AMD, IBM, Qualcomm, Intel, and Nvidia submitted this initial proposal to the Khronos Group.
- **OpenCL 1.1 (June 2010)**
- Enhancements about graphic processing (OpenGL interoperability)
- parallel programming enhancements : flexibility, functionality, and performance
- clEvents to drive and control command execution;
- **OpenCL 1.2 (November 2011)**
- Device partitioning: the ability to partition a device into sub-devices.
- Separate compilation and linking of objects
- Enhanced image support: (OpenGL, DirectX) OpenCL images.
- Built-in kernels (custom devices video encoding/decoding and digital signal processors)

OpenCL 2.0 (November 2013)

- Shared virtual memory
- Nested parallelism
- Generic address space
- Images
- C11 atomics
- Pipes (FIFO)
- Android installable client driver extension

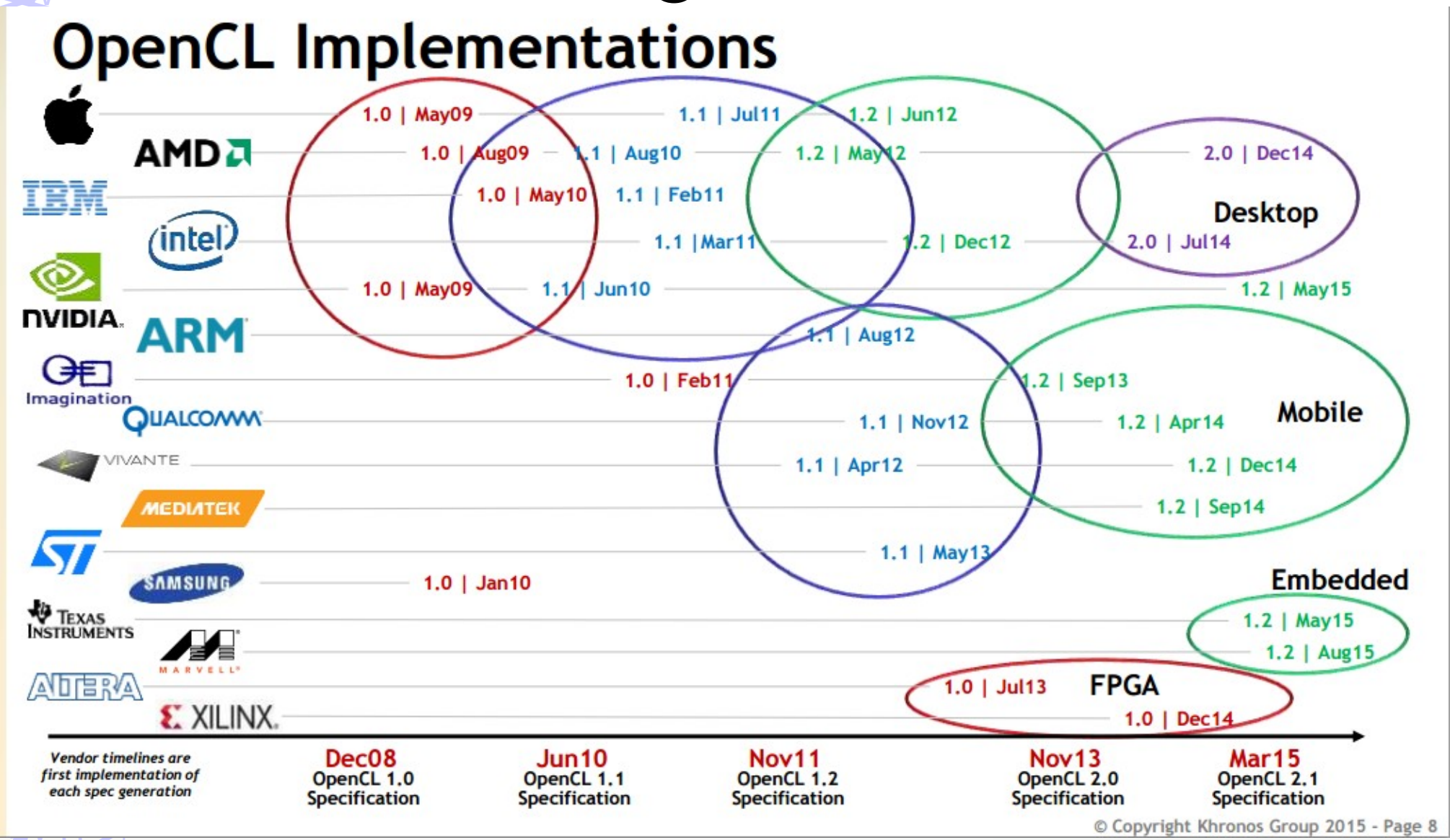
OpenCL 2.1 (November 2015)

- Vulkan and OpenCL 2.1 share SPIR-V :
- Additional subgroup functionality
- Copying of kernel objects and states
- Low-latency device timer queries
- Ingestion of SPIR-V code by runtime
- Execution priority hints for queues
- Zero-sized dispatches from host

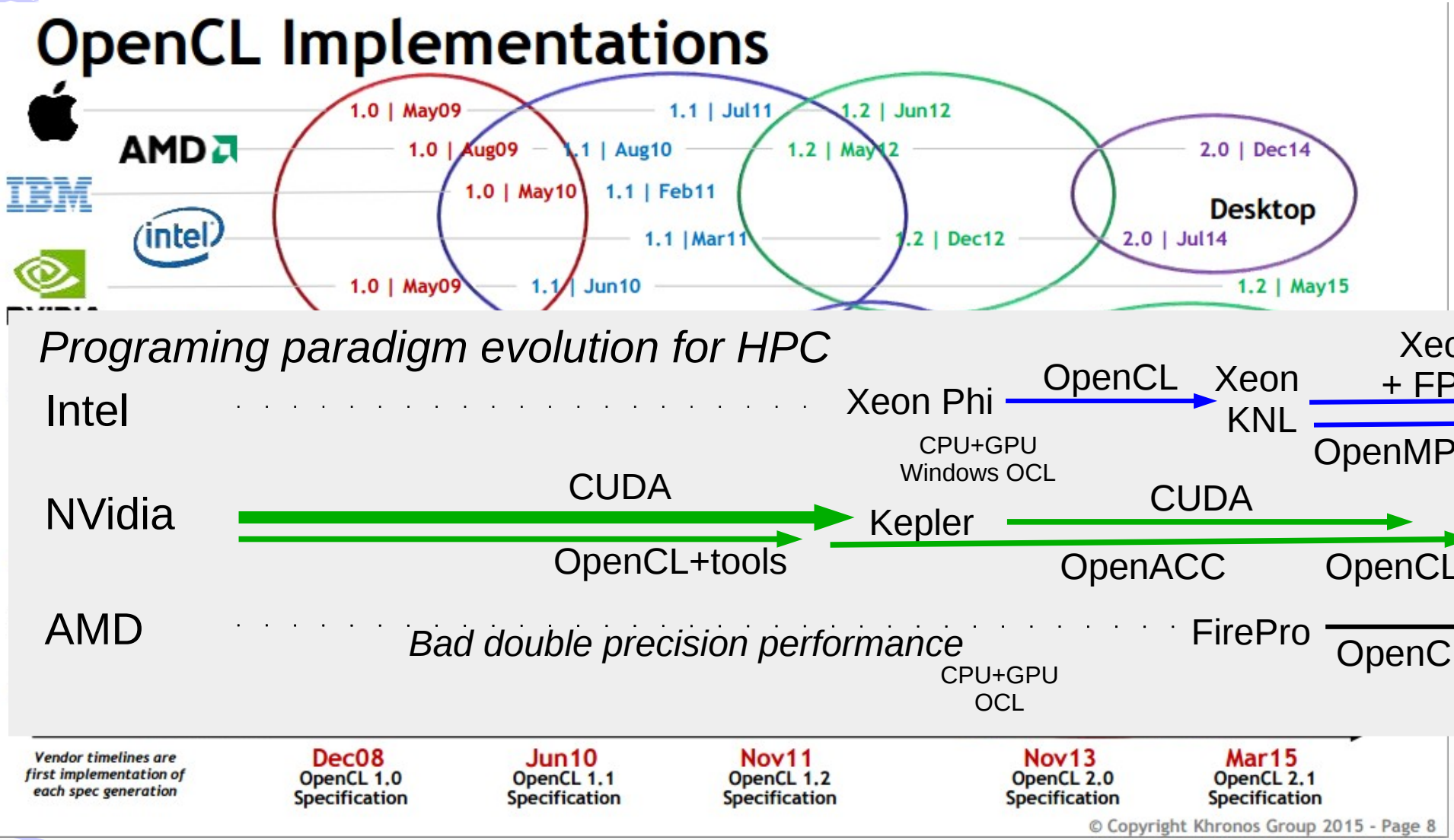
OpenCL 2.2 (March 2016)

"OpenCL 2.2 brings the OpenCL C++ kernel language into the core specification for significantly enhanced parallel programming productivity"

OpenCL Technological choice (2)



OpenCL Technological choice (2)

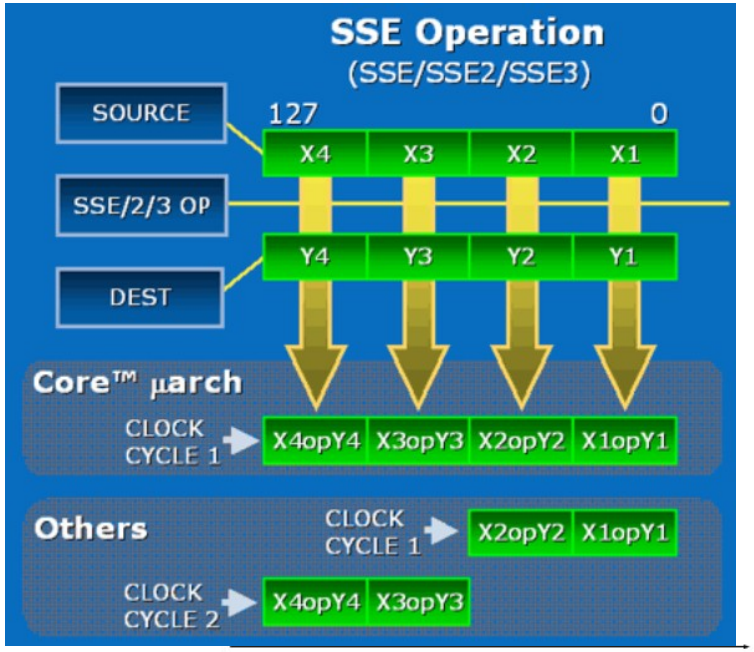


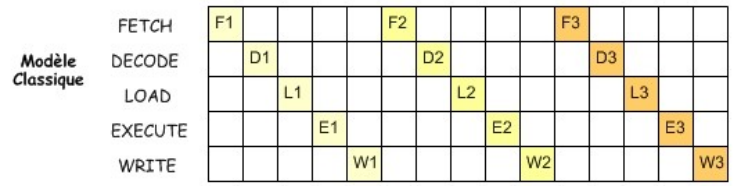
Introduction Vectorization (SIMD)

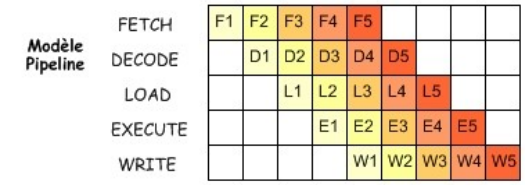
Flynn's taxonomy (CPU instructions): categories SISD, SIMD, MISD, MIMD

- SIMD: Single Instruction Multiple Data (AVX, SSE)

```
# pragma vector always
  for (int i=0; i<n; i++)
    c[i] = a[i] * b[i];
```



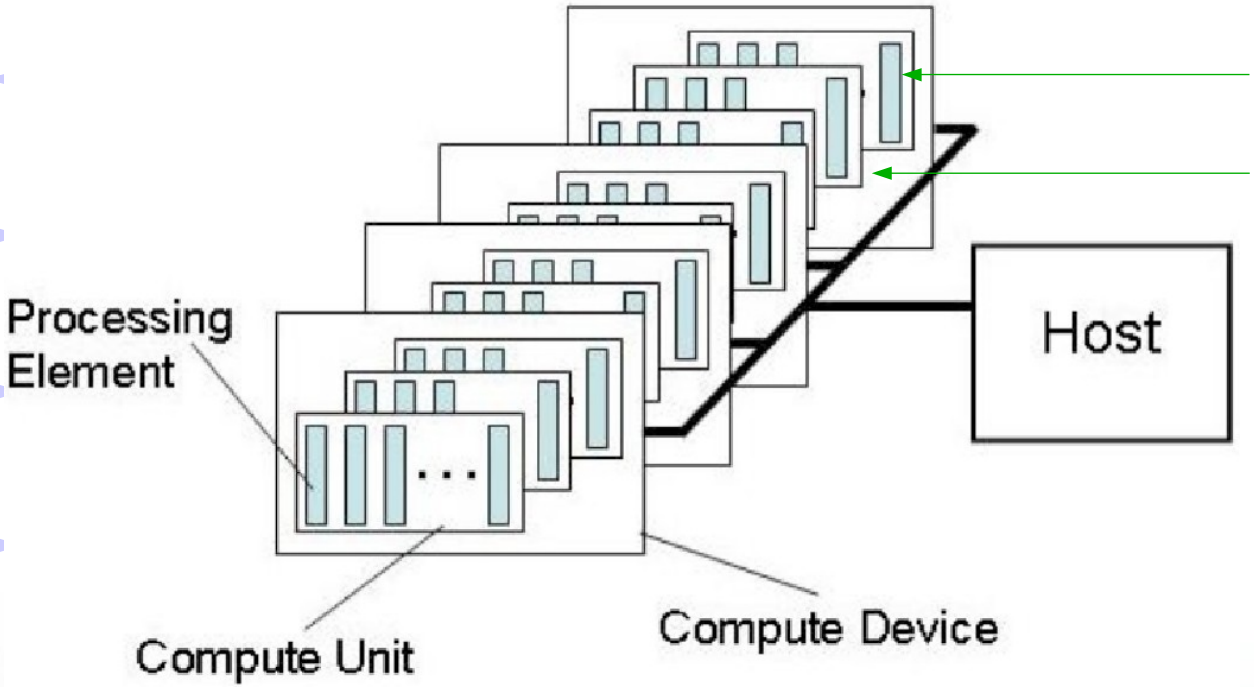




- **Extension to // programs**
- Single Program Multiple Data (SPMD)
- Single Program Multiple Data (MPMD)

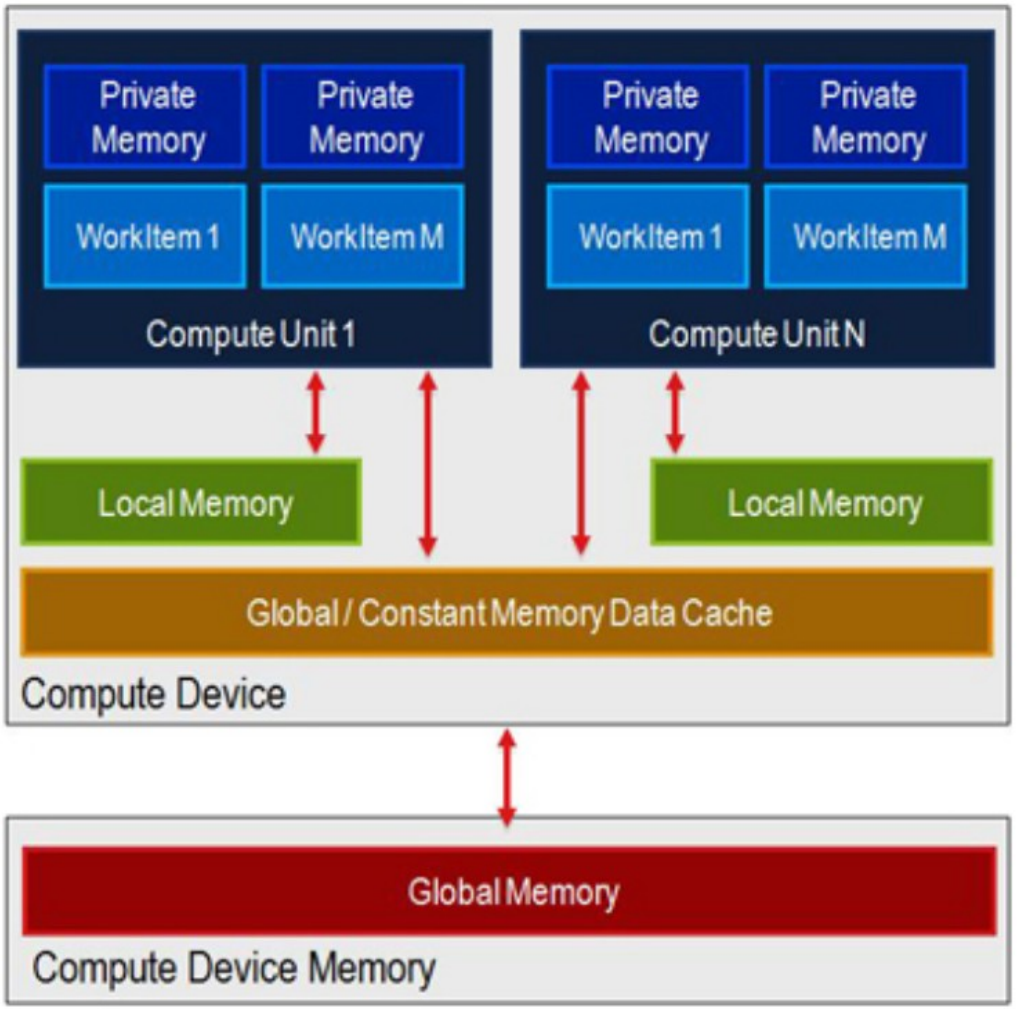
OpenCL Abstract Model (1)

Open **Computing** Language (OCL)



- Handle heterogeneous hardware
- Kernels (C99) → OCL devices
- OCL API → Host/device dialogue (orders, data copies) →
- Hybrid // model:
  - Shared memory model inside one device
  - Distributed memory model: host/device interactions, multi-devices programming

OpenCL Abstract Model (2)



Abstract Processing units

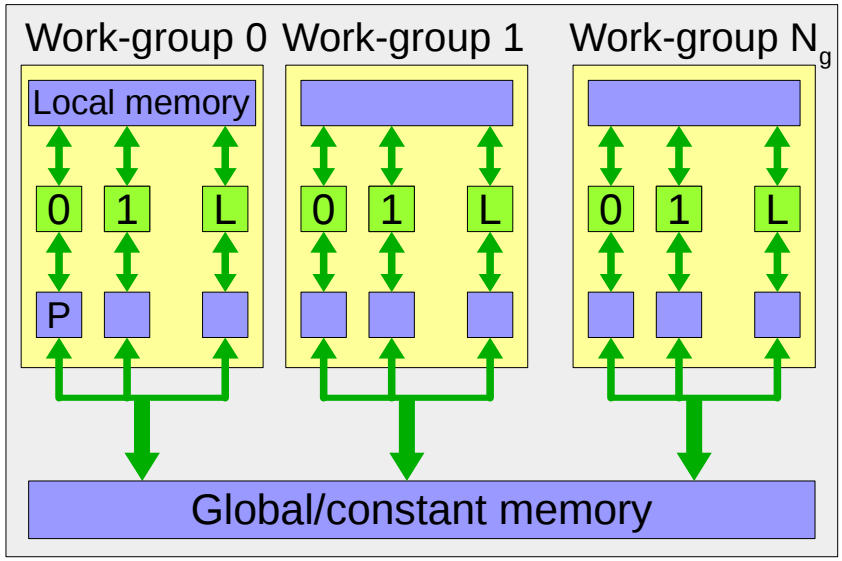
- *Work Item*: computing element
- *Compute Unit*: gather several *Work Item* units
- CPU: core → *Computing Unit*, 1 SIMD element → *Work Item*

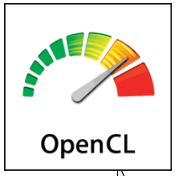
Abstract Hierarchical Memory

- Global memory (shared)
- Data cache (cache L2)
- Local memory (shared in the Compute Unit – cache L1)
- Private memory (registers)

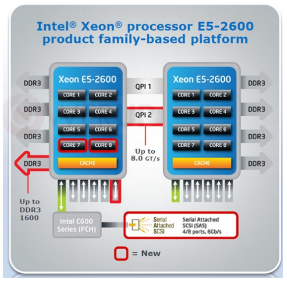
Abstract HW tacking into account parallelism & memory hierarchy

OpenCL Abstract Model (3)

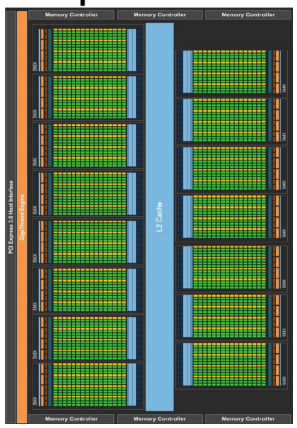




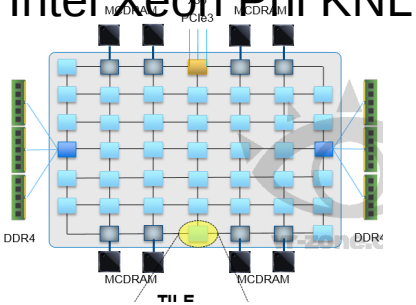
Intel CPU E5-2650



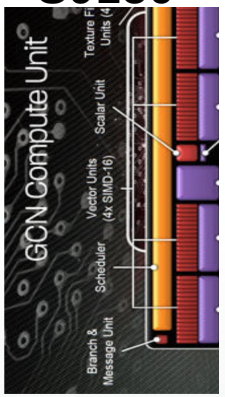
Nvidia Kepler K20



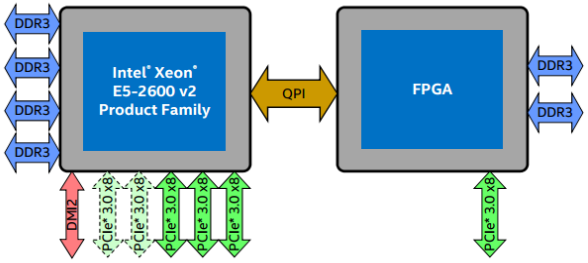
Intel Xeon Phi KNL



AMD FirePro S9150

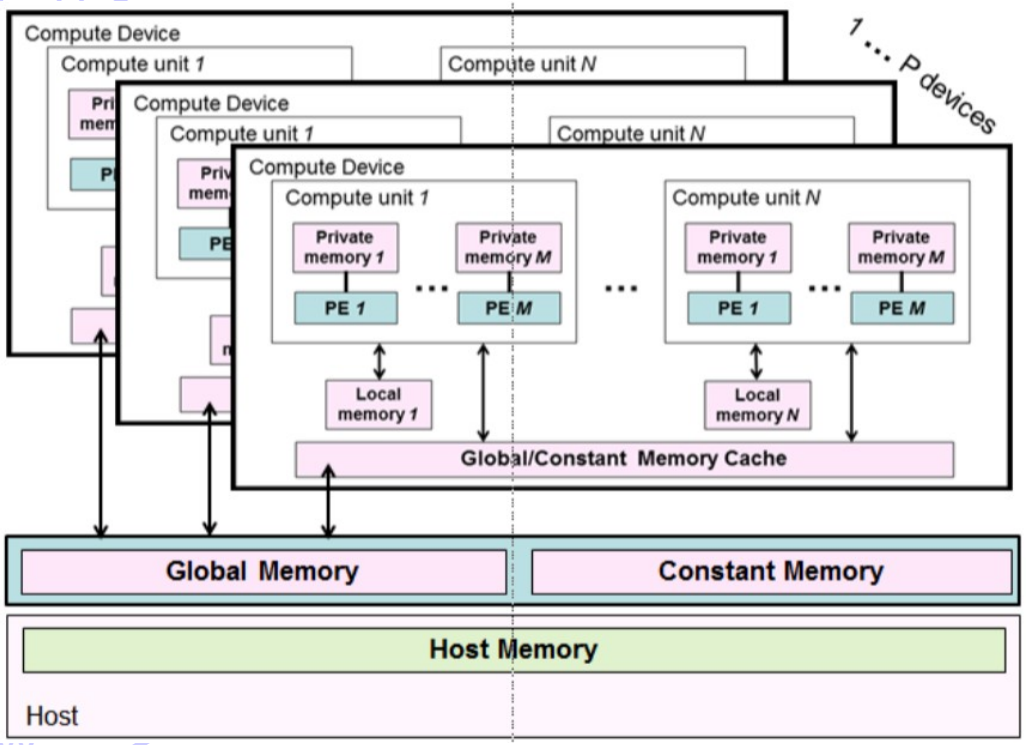


Intel Xeon + FPGA



SW description to target vectorial & parallel hardware

OpenCL Memory Model



- *Global* and *Constant* memories are shared between the one or more devices (within a context)
- *local* and *private* memories are associated with a single device.
- Key words: `__private (default), __local, __global, __constant`

Parallel & vector description, data locality (register, cache, global memory) specification for OCL compiler

OpenCL/CUDA terminology

OpenCL	CUDA
Work Item	*Thread*
Work group	*Thread Block*
`__kernel`	`__global__`
`__global`	`__device__`
`__constant`	`__constant__`
`__local`	`__shared__`
`[__private]`	`[__local__]`

- OpenCL API more general (→ complex)
- *cl.hpp beta* route all OCL calls → corresponding CUDA calls

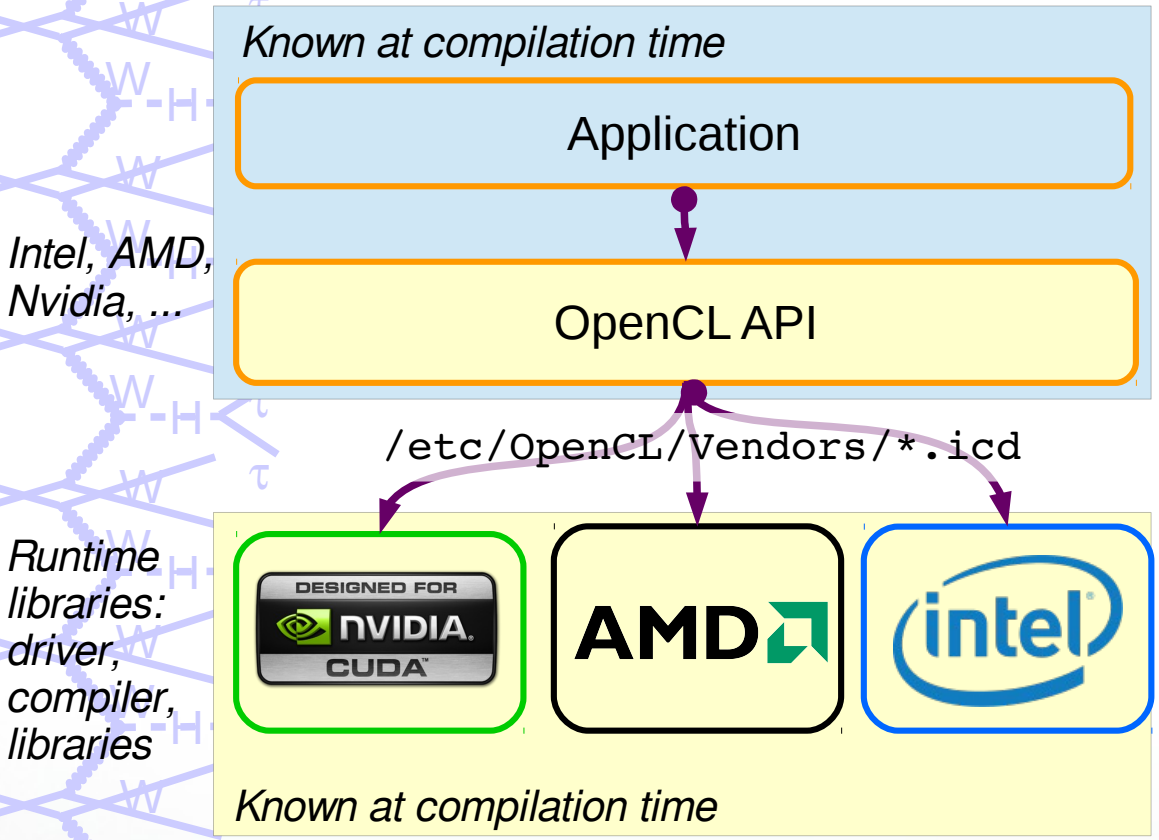
- OpenCL close to CUDA
- OpenCL API mapping CUDA Driver API Run: http://developer.amd.com/

OpenCL	CUDA
`get_num_groups()`	`gridDim`
`get_local_size()`	`blockDim`
`get_group_id()`	`blockIdx`
`get_local_id()`	`threadIdx`
`get_global_id()`	*deduced*
`get_global_size()`	*deduced*

Easy to convert OpenCL to CUDA

http://developer.amd.com/tools-and-sdks/opencl-zone/opencl-resources/programming-in-opencl/porting-cuda-applications-to-opencl/

OpenCL API & kernels

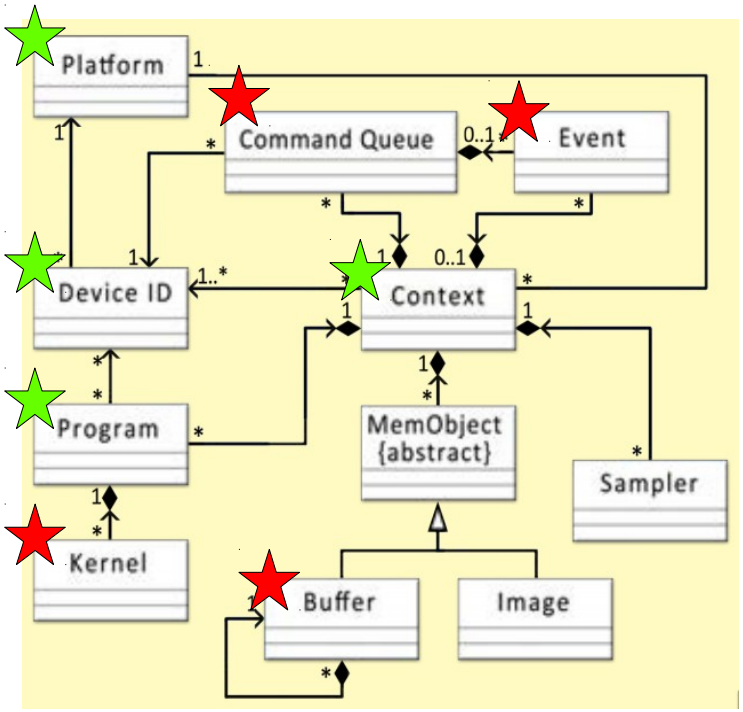


Library selection mechanism, ex: kernel compilation

Two parts to distinguish:

- **API** part → OCL SDK environment (hands-on → OpenCL Intel SDK)
- **Kernel code** part
- The compiler, libraries... are selected at runtime: compiler/library are located thanks to `/etc/OpenCL/Vendors/*.icd` files
- One .icd file→ one OCL Platforms:
  - llrgrcl01-02, 2 platforms: NVidia, Intel
  - Llrgrcl03-04, 1 platforms: Intel

OpenCL API
Abstract classes



- Images, OpenGL … not presented here

C++ Wrapper

- Discover platforms (selection)

  `cl::Platform, cl::Device`
- Computation context and kernel building:

  `cl::Context, cl::Program, cl::Kernel`
- Device memory allocations:

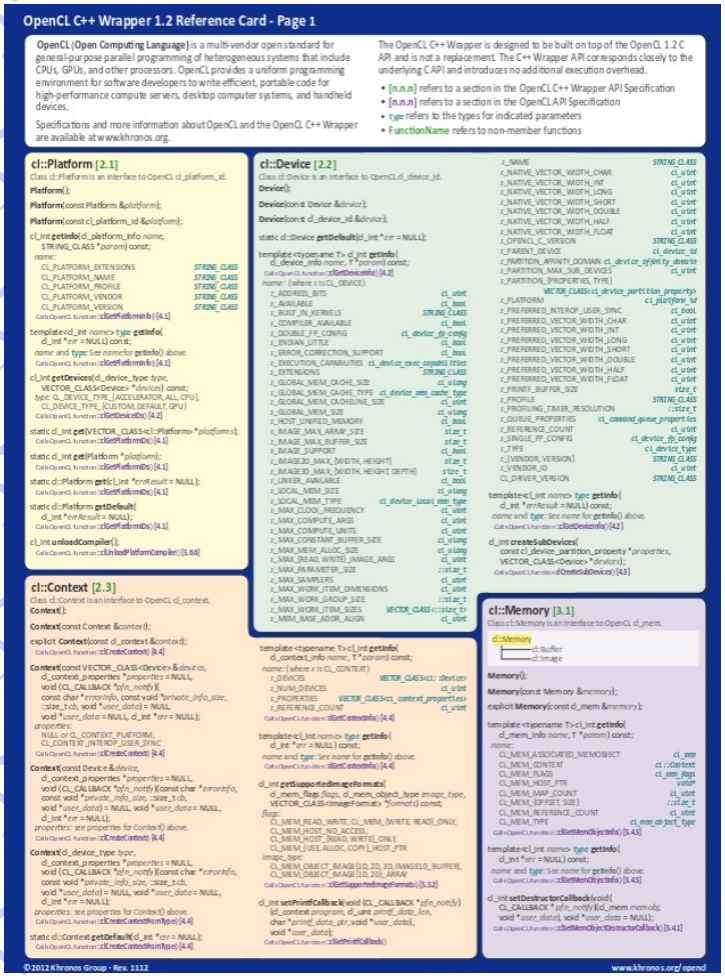
  `cl::Buffer`
- Computation:

  `cl::CommandQueue, cl::Kernel, cl::Buffer, cl::Event`

Few OpenCL classes which deals with computation

OpenCL API
C++ API

OpenCL C++ Wrapper 1.2 Reference Card (6 p.)



https://www.khronos.org/files/OpenCLPP12-reference-card.pdf

- API is rich of functions/methods (getting information on configurations & object status).
- Numerous combinations: we will see one way to interact with API.
- The user manages the error code or enable OCL exceptions

```
# define __CL_ENABLE_EXCEPTIONS
# include <CL/cl.hpp>
```

OpenCL API
Platforms & Devices (1)

Main platform and device discovery methods

- Querying available platforms
  static cl_int **cl::Platform::get**(vector<cl::Platform> **platforms*);
- List of available devices on **a** platform.

  cl_int **cl::Platform::getDevices**(cl_device_type *type*, vector<cl::Device> * *devices*);

  type: CL_DEVICE_TYPE_{ACCELERATOR, ALL, CPU, CUSTOM, DEFAULT, GPU}

One .icd file → one OCL Platforms:
- llrgrcl01-02, 2 platforms: NVidia, Intel
- llrgrcl03-04, 1 platforms: Intel

cl_int **clGetPlatformIDs**(cl_uint nb_entries, cl_platform_id *platforms, cl_uint *nb_platforms)

OpenCL API
Platforms & Devices (2)

Get platform & device features:

cl_int cl::Platform::getInfo(cl_platform_info name, string *param);

name: CL_PLATFORM_{EXTENSIONS, NAME, PROFILE, VENDOR, VERSION}

- template <typename T>

  cl_int cl::Device::getInfo(cl_device_info name, T * param)
- name = CL_DEVICE_{VENDOR, NAME, OPENCL_C_VERSION, EXTENSIONS, MAX_WORK_ITEM_SIZES , ...}

OpenCL API Context

Creating an execution context (command-queues, memory, programs and kernels) for a (sub-)set of devices belonging to the same platform

- cl::Context::Context(vector<Device>& *devices*, cl_context_properties * *properties* = NULL, void (CL_CALLBACK * *pfn_notify*)(…) = NULL, cl_int * *err* = NULL)

  properties: CL_CONTEXT_PLATFORM, CL_CONTEXT_INTEROP_USER_SYNC, CL_GL_CONTEXT_KHR, …
- template <cl_int name> typename … cl::Context::getInfo()

OpenCL API
Device memory allocation (Buffer)

Class cl::Buffer : public Memory

The constructor

cl::Buffer::Buffer(const Context& *context*,
cl_mem_flags *flags*, ::size_t *size*,
void **host_ptr* = NULL, cl_int **err* = NULL)

flags: CL_MEM_{READ_WRITE, WRITE_ONLY, READ_ONLY},
CL_MEM_{USE_HOST_PTR, ALLOC_HOST_PTR, COPY_HOST_PTR}

size: the size (in bytes) of the buffer

host_ptr: buffer already allocated by user application on host

- Remark: no device specified

OpenCL API
Building Kernels

Program

- One of the constructors from sources or binaries

  cl::Program::Program(const Context& context, const Sources& sources, cl_int *err= NULL)

  sources: vector<std::pair<const char*, ::size_t> > Sources
- Compile and link
  cl_int cl::Program::build(const vector<Device>& devices, const char *_Options_ = NULL, (CL_CALLBACK * _pfn_notify_) ... = NULL)
  Options: compiler options

Kernel

- cl::Kernel::Kernel(const Program& program, const char *name, cl_int *err = NULL)
  name: kernel name (`__kernel` qualifier)
- template <typename T>
  cl_int cl::Kernel::setArg(cl_uint _index_, T _value_)
  value: argument of type int, long int, double, …, cl::Buffer, cl::Local(m)
  index: number in the argument list (0..n-1)
-

OpenCL API
Command Queues

Command queue of orders to **a** device (copies, launching kernel, waiting)

```
cl::CommandQueue::CommandQueue(
        const Context& context, const Device& device,
        cl_command_queue_properties properties = 0,
         cl_int * err = NULL)
```

device: a device of the context

properties:

CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE,
CL_QUEUE_PROFILING_ENABLE

OpenCL API
Host/Devices copy

- **Device → host copy**

```
cl_int cl::CommandQueue::enqueueReadBuffer( const Buffer& buffer,
        cl_bool blocking_read, ::size_t offset, ::size_t size, const void *ptr,
        const vector<Event> *events = NULL, Event *event = NULL)
```

- **Host → device copy**

```
cl_int cl::CommandQueue::enqueueWriteBuffer( const Buffer& buffer,
        cl_bool blocking_write, ::size_t offset, ::size_t size, const void * ptr,
        const vector<Event> *events = NULL, Event *event = NULL)
```

buffer: allocated in a cl::context
blocking_read: wait the end of the copy
offset: from *buffer* location in the device (generally 0)
size: of the data to copy (in bytes)
ptr: memory location (in the host memory !)
events: event list to wait before to perform the copy (if 0 no event dependency)
event: event set completed when the copy is finished (0 no event updated)

OpenCL API CommandQueue & Event

CommandQueue synchro.

- cl_int **cl::CommandQueue::Finish**()

  Blocks until all previously commands are issued

- cl_int **cl::CommandQueue::Flush**()

  Only guarantees that all commands is submitted. There is no guarantee that they will be complete

- cl_int **cl::CommandQueue::enqueueBarrier**()

  Enqueues a barrier operation

Event synchronizations

- cl_int **cl::Event::wait**()

  Waits on the host thread for the command associated with the event to complete

- static cl_int **cl::Event::waitForEvents**(
  const vector<Event> &*events*)

  Waits on the host thread for the command associated with the event list to complete

OpenCL API Kernel (1)

Kernel submission

```
cl_int cl::CommandQueue::enqueueNDRangeKernel(
        const Kernel& kernel, const NDRange& offset,
        const NDRange& global, const NDRange& local,
        const vector<Event> *events = NULL,
        Event *event = NULL)
```

kernel: in the same Context than the CommandQueue
offset: shift in the work-item indexing
global: global size of the indexing (number of work-items). Must be a multiple of *local*
local: work-size (< device capacity, often 1024)
Automatic if cl::NULLRange
events, *event*: same as for Host ↔ Device copy

NDRange (specifies data dimensions)
1D constructor cl::NDRange::NDRange (::size_t s0)
2D constructor cl::NDRange::NDRange (::size_t s0, ::size_t s1)
3D constructor cl::NDRange::NDRange (::size_t s0, ::size_t s1, ::size_t s2)

OpenCL API
Kernel (2)

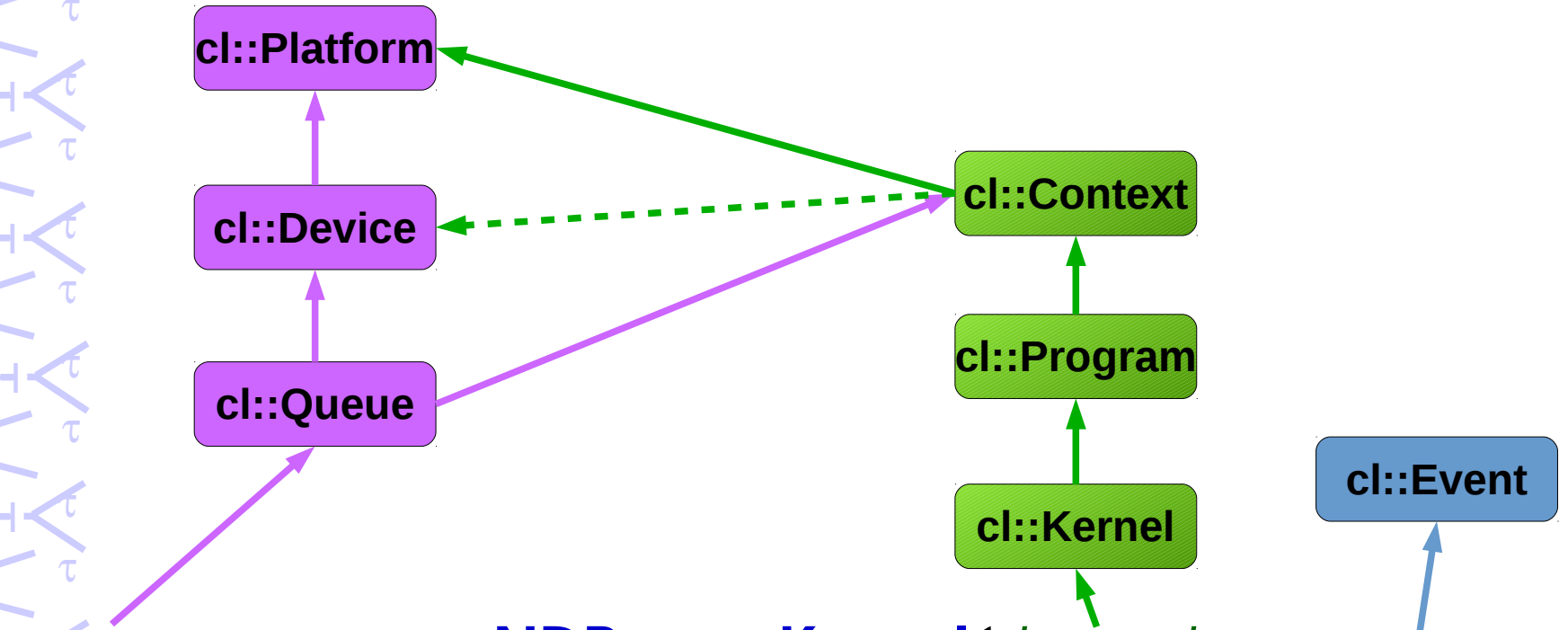
- cl_int **cl::CommandQueue::enqueueTask**(const Kernel& *kernel*,
  const vector<Event> **events* = NULL, Event **event*= NULL)

  Enqueues a command to execute a kernel on a device. The kernel is executed using a single work-item.

- cl_int **cl::CommandQueue::enqueueNativeKernel**(
  void (*user_func) (void *), std::pair<void*, ::size_t> args,
  const vector<Memory> *mem_objects= NULL,
  const vector<const void *> *mem_locs = NULL,
  const vector<Event> *events = NULL, Event *event = NULL)

  Enqueues a command to execute a native C/C++ function not compiled using the OpenCL compiler.

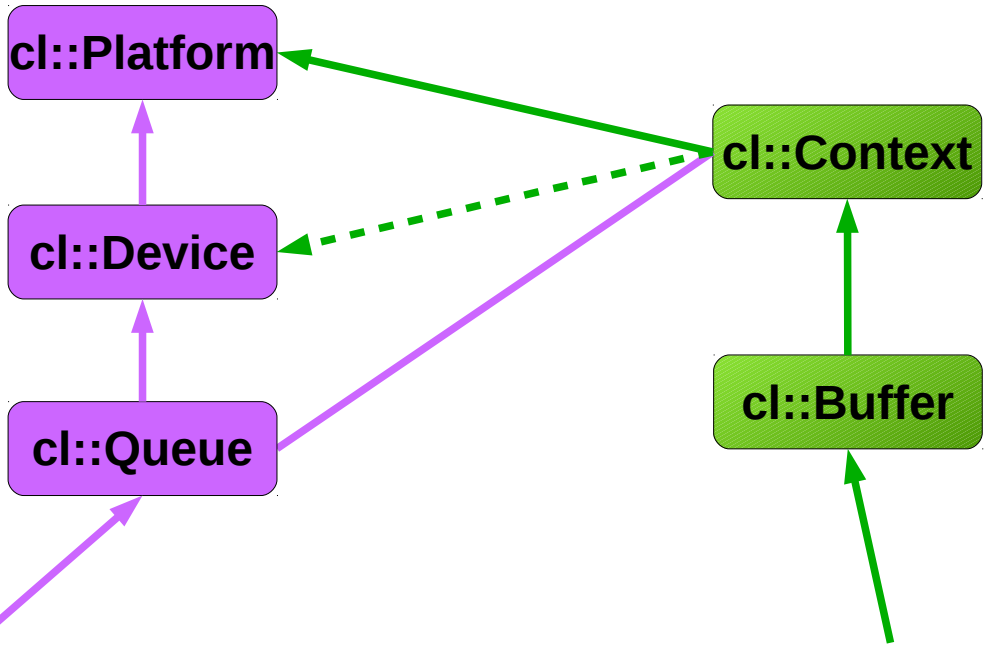
OpenCL API
Summarize graph (1)



queue.**enqueueNDRangeKernel**(*kernel*,
offset, *global*, *local*, *events*, *event*)

queue & *kernel* must be coherent

OpenCL API
Summarize graph (2)



queue.**enqueueReadBuffer**(buffer,
CL_FALSE, offset, size, ptr, events, event)

OpenCL Kernels Rules

__kernel **void** dotProduct(__global char *toto, __global double *A,
__global double *B, long int *vectorSize*);

- Memory location __global, __constant, or __local qualifiers
- Attribute kernels qualifiers (work-size, memory access, ...)

Main C99 restrictions

Restricted environment but highly compatible with performances could evolve in the future (more and more the OS/hardware/compiler evolve on the accelerator system)

- No reference (pointer) to variables in the data (pb of mapping addresses)
- Pointers to functions are not allowed
- The library functions defined in the C99 standard headers (no malloc)
- `static` memory qualifier forbidden in a kernel (static variables → allocated by OpenCL)
- `complex` type OpenCL 1.2, not in OpenCL 1.1 (NVidia ...)
- int printf(constant char *restrict format, ...) not in OpenCL 1.1 (NVidia ...)

OpenCL kernels
Work-Item built-in functions

API side

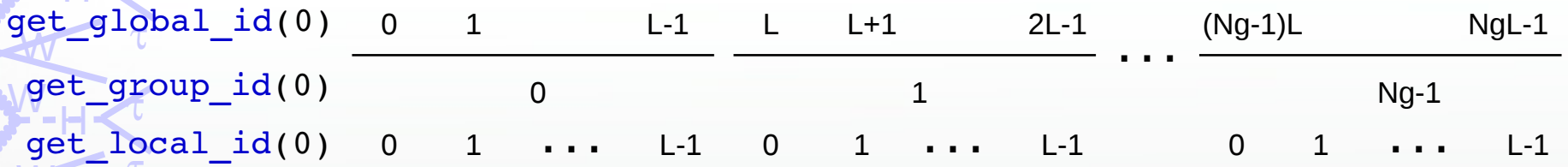
```
cl_int cl::CommandQueue::enqueueNDRangeKernel(
    const Kernel& kernel,
    const NDRange& offset,
    const NDRange& global,
    const NDRange& local,
    const vector<Event> *events,
    Event *event )
```

Kernel Side

- Fixed at enqueueNDRangeKernel call:

  get_work_dim(dim),
  get_global_offset(dim),
  get_global_size(dim),
  get_local_size(dim),
  get_num_groups(dim)

- Variable at the kernel run-time
  with *local* = work-group size
  global = *Ng*local*



OpenCL kernels
Built-in functions

Lot of built-in function: common, integer, math, …

Preprocessor Directives & Macros

- # pragma OPENCL FP_CONTRACT : {ON, OFF, DEFAULT}
- # pragma OPENCL EXTENSION cl_khr_fp64 : enable
- Defined macro:
  __FILE__, __LINE__
  __OPENCL_VERSION__, __CL_VERSION_X_Y__

… … …

OpenCL kernels Synchronization functions

Classical in HPC terminology (MPI, OpenMP)

Barrier

- void **mem_fence**(
  cl_mem_fence_flags *flags*)
  Loads and stores preceding the mem_fence will be committed before any loads and stores following the mem_fence
  flags: CLK_{LOCAL,GLOBAL}_MEM_FENCE
- void **barrier**(cl_mem_fence_flags *flags*)
  Wait all work-items in **a work-group** reach the barrier() before to continue.
  Must be encountered by all work-items in a work-group

Atomic functions

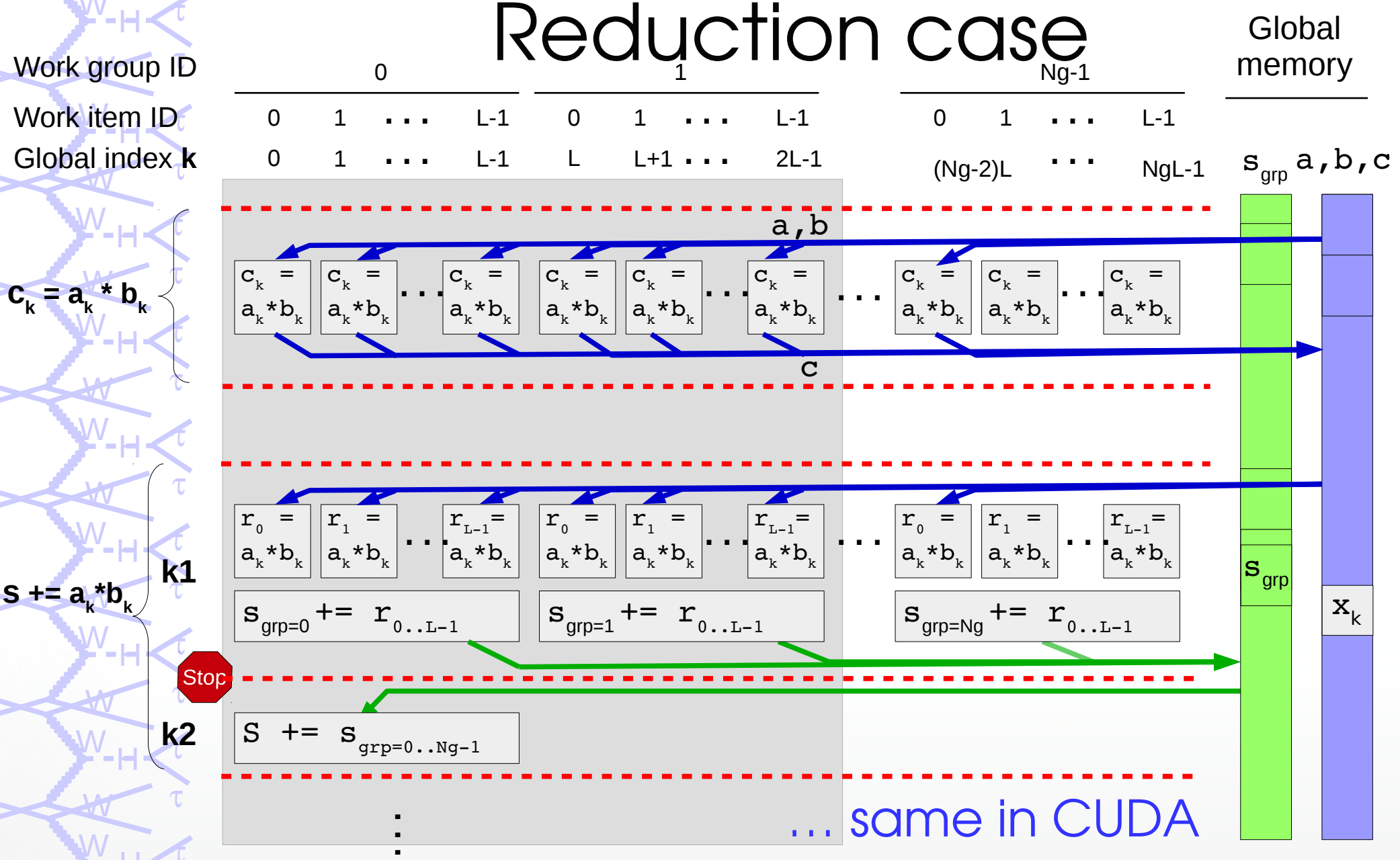
type **atomic_*op***(
memory_location *type* value,
type privateValue)

op: inc, dec, add, sub, min, max, or, and, xor, xchg
type: especially integer types
memory_location: {__global, __local}

Performs a group of instructions as one (unspittable) instruction

Prefetch & Async Copies (Global ↔ Local Memory)
not detailed here

OpenCL Reduction case



OpenCL Optimization

Main ideas

- Try to keep sources (kernel) unchanged
- Portables optimizations
- Kernel workload heavy (hide latencies)
- Avoid global memory access
- Memory accesses (contiguous)

Not treated here

- Specific optimization to specific hardware

High level & portable optimizations

OpenCL & MPI

Part 2: OneDevice

- Code $e^+e^- \rightarrow \gamma\gamma\gamma$: parallelism
- Random Number Generator
- OpenCL implementation ($e^+e^- \rightarrow \gamma\gamma\gamma$)
- GridCL Platform
- Hands on

M. Rubio-Roy, A. Sartirana, F. Thiant

OneDevice
Code $e^+e^- \rightarrow \gamma\gamma\gamma$ & parallelism

```
// Initilization
// rambo, ME, RNG, ...
for ( ev=0; ev < run->nbrOfEvents; ev++) {
  //
  // Reset Matrix elements
  resetME2( &ee3p );
  //
  // Event generator
  eventWeight = generateRambo ( &rambo,
                        &rngstate, outParticles, 3, run->ETotal );
  eventWeight = eventWeight * run->cstVolume;
  //
  // Sort outgoing photons by energy
  sortPhotons( outParticles );
  //
  // Spinor inner product, scalar product and
  // center-of-mass frame angles computation
  computeSpinorProducts( &ee3p.spinor, ee3p.momenta );
  computeScalarProducts( &ee3p );
  //
  if ( selectEvent( &pParameters, &ee3p  ) ) {
    computeME2( &ee3p, &pParameters, run->ETotal );
    updateStatistics( &statistics, &pParameters, &ee3p,
                                                 eventWeight );
    evSelected++;
  }
}
statistics.nbrOfSelectedEvents = evSelected;
// Finalize ....
```

Main loop

Not specific to OpenCL

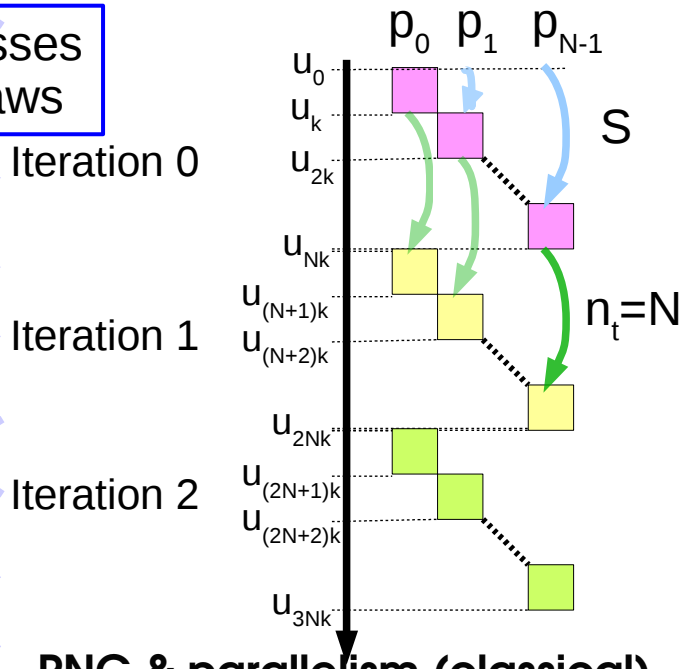
- Events are independent
- *Work-sharing*: events
- *Private data*: dealing with one event, even the RNG status
- *Shared data*: statistics
- Reduction to sum the proba, variance, …

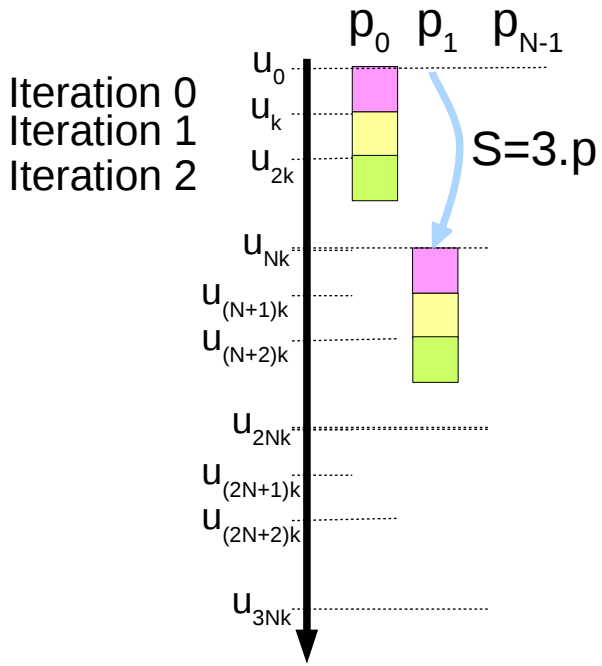
Simple study which cannot be done by compilers

OneDevice Parallelism & RNG

Random sequence distributed among concurrent processes

N // processes
3 RNG draws





RNG & parallelism (classical)

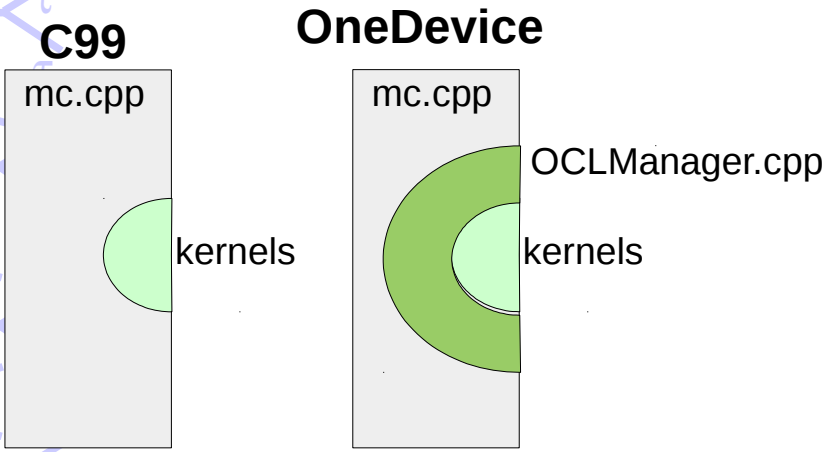
- To avoid correlation, shifts are performed (S, N iterations).
- With $n_t = k \times N$, N number of processes
- Very large $n_t \sim 10^4 \times 2d \times 10^2 \times N_{integrals}$

Optimization:

- $n_t = 0$
- No correlation inside an event

Recent RNG (mersenne twister) include a key which facilitate //

OneDevice OpenCL implementation (1)



C99 — **Kernels** — **GSL**
MC — **MC**
mc.cpp

OneDevice — **Kernels** — **GSL**
config.h
MC — **MC**
mc.cpp — statistics.cpp
OCL

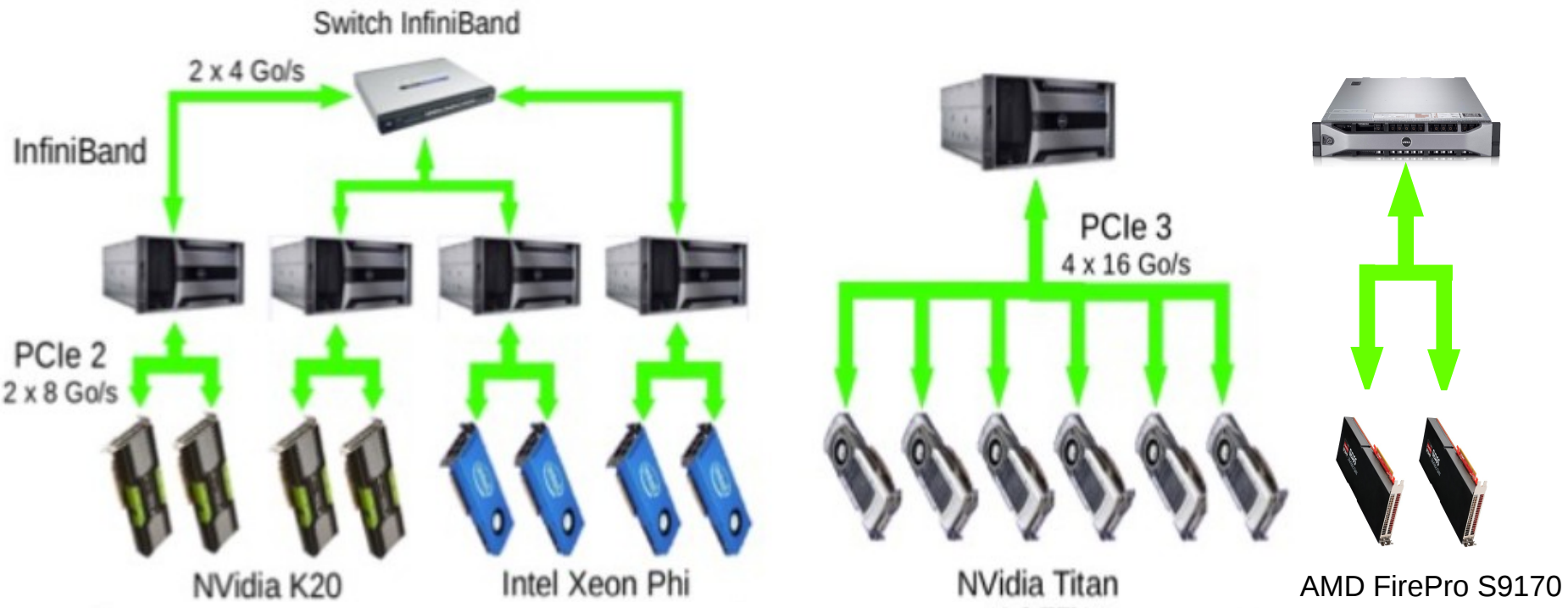
C99 initial code

`complex → cmplx_t`

C99 → OneDevice

- kernels unchanged
- adding Kernels/OCL directory for API part
- Modification of Kernels/MC/stastistics.c for reduction part.

OneDevice
GridCL platform (1)



Each node

- 2 x Intel E5-2650: 2 GHz,16 cores, with AVX (4 doubles), hyperthreading 32 cores
- 64 Go memory

Interconnection
switch InfiniBand

OneDevice
GridCL: NVidia K20, Titan (2)

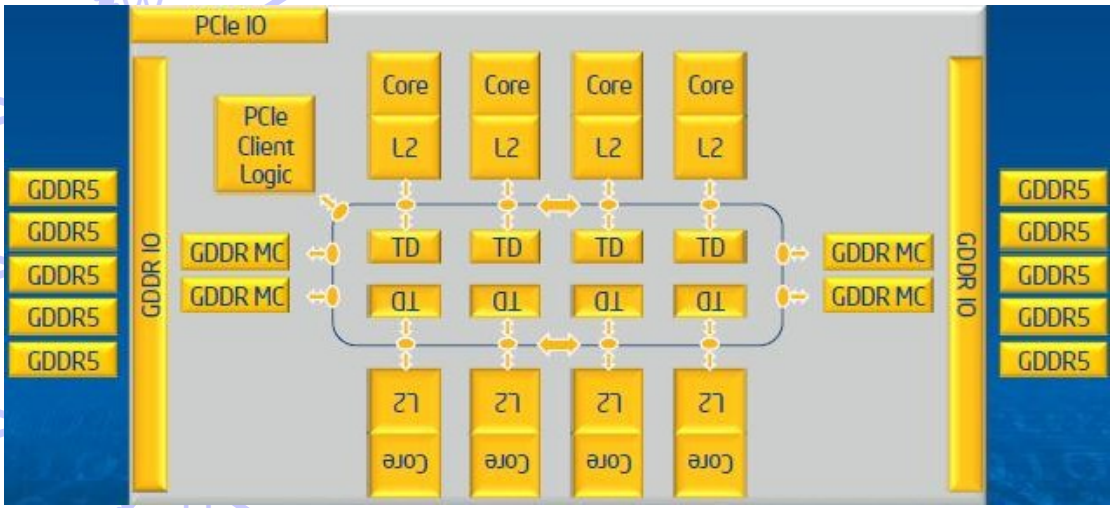
- Kepler GK110 , 13 processors (SMX), 192 *cores*/threads → *2496 cores, 706 MHz, 225 Watt*
- Work-Group divides in Warp (32 threads)
- Each SMX deals with up to 64 warps
- 255 Registers per threads (32-bit)
- Local memory (shared mem.) limited to 48 kB (L1 cache)
- 5 Gb memory, BW 208 Gb/s

- 1, 17 TFlops (3.52 SP) Peak performance: 13*64*/ 0.706 x10^6
- Coalescent memory access
- Titan similar, Pascal → 5 TFlops

	FERMI GF100	FERMI GF104	KEPLER GK104	KEPLER GK110	KEPLER GK210
Compute Capability	2.0	2.1	3.0	3.5	3.7
Threads / Warp	32				
Max Threads / Thread Block	1024				
Max Warps / Multiprocessor	48		64		
Max Threads / Multiprocessor	1536		2048		
Max Thread Blocks / Multiprocessor	8		16		
32-bit Registers / Multiprocessor	32768		65536		131072
Max Registers / Thread Block	32768		65536		65536
Max Registers / Thread	63			255	
Max Shared Memory / Multiprocessor	48K				112K
Max Shared Memory / Thread Block	48K				
Max X Grid Dimension	2^16-1		2^32-1		
Hyper-Q	No			Yes	
Dynamic Parallelism	No			Yes	

http://international.download.nvidia.com/pdf/kepler/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf

OneDevice
GridCL: Intel Xeon Phi (3)



- Intel® Many Integrated Core (MIC)
- Intel Xeon Phi, Knight Corner, 60 true cores, 4 threads / *core* → 240 *cores,*
- *Clock 1.053 GHz, 225 Watt*
- 1 VPU per core: 512-bit SIMD instruction set, 8 double-precision (DP) operations per cycle
- Fused Multiply-Add (FMA) instructions → 16 DP / cycle

- 128 Vector Registers per core (128x512-bit)
- Peak Perf. (DP):
  = 16 (VPU) x 60(cores) x 1.053 (clock)
  = 1.011 Tflops
- 8 Gb memory, BW 320 Gb/s

Intel® Xeon Phi™ Coprocessor Family Reference Table

SKU #	Form Factor, Thermal	Peak Double Precision	Max # of Cores	Clock Speed (GHz)	GDDR5 Memory Speeds (GT/s)	Peak Memory BW	Memory Capacity (GB)	Total Cache (MB)	Board TDP (Watts)	Process
SE10P	PCIe Card, Passively Cooled	1073 GF	61	1.1	5.5	352	8	30.5	300	
SE10X	PCIe Card, No Thermal Solution	1073 GF	61	1.1	5.5	352	8	30.5	300	
5110P	PCIe Card, Passively Cooled	1011 GF	60	1.053	5.0	320	8	30	225	22nm
3100 Series	PCIe Card, Actively Cooled	>1 TF	Disclosed at 3100 series launch (H1'13)		5.0	240	6	28.5	300	
	PCIe Card, Passively Cooled	>1 TF			5.0	240	6	28.5	300	

https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner

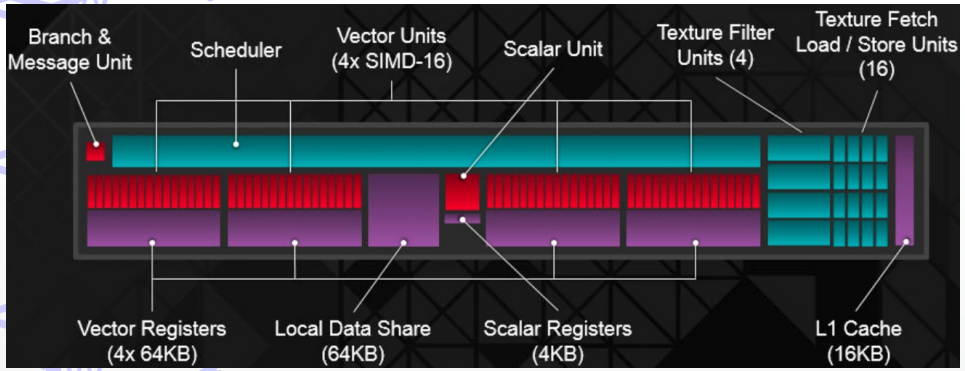
OneDevice
GridCL: AMD Firepro S9170 (4)

	FirePro S9170	FirePro S9150	FirePro S9100	FirePro S9050
Cooling	Passive	Passive	Passive	Passive
Stream Processors	2816	2816	2560	1792
OpenCL™ Support	2.0	2.0	2.0	1.2
GPU Compute (SP)	5.24 TFLOPS	5.07 TFLOPS	4.22 TFLOPS	3.23 TFLOPS
GPU Compute (DP)	2.62 TFLOPS	2.53 TFLOPS	2.11 TFLOPS	806 GFLOPS
Memory Size	32 GB	16 GB	12 GB	12 GB
Memory Bandwidth	320 GB/sec	320 GB/sec	320 GB/sec	264 GB/s
Memory Interface	512-bit	512-bit	512-bit	384-bit
Memory ECC	Y	Y	Y	Y
PCI Express® Bandwidth	32 GB/sec	32 GB/sec	32 GB/sec	32 GB/sec
TDP	275W	235W	225W	225W

https://www.amd.com/Documents/firepro-s9150-datasheet.pdf

llrgrcl05.in2p3.fr

- Tahiti/Hawaii (Core next generation)
- 2.62 Tflops (DP)
- 32 GB GDDR5 memory
- TDP 275 W



- 44 processors
- 64k 32-bit registers / proc.
- Flops = 64 * 64 (SP) * freq. ?

OneDevice Hands on (1)

Two kernels

- Cumulate the event contribution: `runEventKernel`
- The kernel take into account all the events: 10^6 ev → global size
- Reduction (finish) `collectStastisticsKernel`
- All variables to reduce are done with two (kernel) functions:

```
$ hg clone
https://hyhg@bitbucket.org/hyhg/opencl-mpi
# Build
$ cd opencl-mpi/OneDevice/MC
$ . ../../Env/gridcl.env
$ (cd ..; make clean; make)
# Increasing the stack limit
$  ulimit -s unlimited
$ ./mc
# Change platform & device
# in mc.cpp ~ line 68:
OCLManager OCLMgr( 0, 0 );
```

```
reduceInWorkGroupType( Type pData, __global Type *gData)
reduceBetweenWorkGroup( __global Type *gData, nbrOfWorkGroups)
```

OneDevice Hands on (2)

Compile the application (API & kernel parts)

To help:

- "Question" in the source missing code, bugs, parallelism, optimization
- kernel compilation: `make kernels` use the Intel OCL compiler `ioc64`

High level optimizations

- Worksize
- Kernels
- Ideas to optimize (the implementation not optimal)
- Optimize reduction

Tools

OneDevice Hands on (3)

- Focus on using OpenCL API
- Not optimal implementation
- Optimization: find best workgroup sizes

- Following performances are obtain with dedicated machines.

	C++ 1-core	OCL 16-cores	OCL K20	OCL Titan	OCL X. Phi	AMD FirePro
Time/ev (µs)	2.45	0.160	0.112	0.0870	0.157	
Speed-up	1	15.3	21.9	28.2	15.6	

OneDevice Correction (4)

As expected the performance are not satisfactory

	C++ 1-core	OCL 16-cores	OCL K20	OCL Titan	OCL X. Phi	AMD FirePro
Time/ev (µs)	2.45	0.160	0.112	0.0870	0.157	
Speed-up	1	15.3	21.9	28.2	15.6	

OpenCL & MPI

Part 3: Multi-Devices

- Code $e^+e^- \rightarrow \gamma\gamma\gamma$: parallelism
- Random Number Generator
- OpenCL implementation ($e^+e^- \rightarrow \gamma\gamma\gamma$)
- GridCL Platform
- Hands on
- cl.hpp & nvprof

Multi-Devices Context

Unique changes in **Kernels**

```
_kernel void runEventsKernel( ...
      unsigned long int nbrOfIterations ) {
...
for (ev=0; ev < nbrOfIterations; ev++) {
  evWeight = generateRambo ( rambo, &rng,...);
  evWeight = evWeight * run->cstVolume;
  // Sort outgoing photons by energy
  sortPhotons( outParticles );
  // Spinor inner product, scalar product and ...
  computeSpinorProducts( &ee3p.spinor,...);
  computeScalarProducts( &ee3p );
  // Reset Matrix elements
  resetME2( &ee3p );
  // Select Event
  GID = get_gloabl_gid(0)*nbrOfIterations+ev
  If((GID < nbrEventsToProcess)
     && selectEvent( pParameters, &ee3p)) {
    computeME2( &ee3p, pParameters, ...);
    evSelected += 1;
  }
  updateStatisticsInWorkgroup( rStorage,... );
}
ReduceInWorkGroupLongInt( evSelected,...);
// Save the RNG status
```

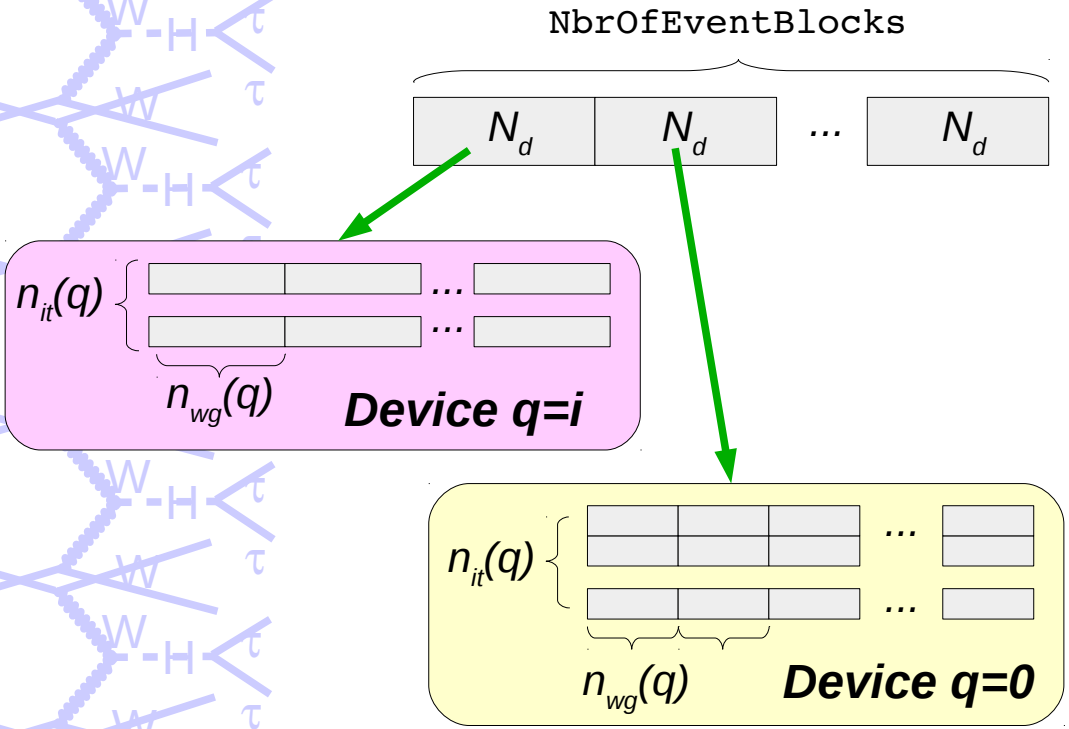
Speed up with **OneDevice**

Problems / Solutions

- Not enough work for Devices
  → several events proceeded in `runEventKernel` (optimization)
- The efficiency of the devices not homogeneous
  → launch kernels up to complete all events

  → Use asynchronous OCL mechanisms
- Adjust RNG initializations
- Global reduction done once
- Run the **same** kernels
- Reduce the memory for RNG status
- Increase Number of event

MultiDevices
Parallelism between devices



- `OneDevice` not optimized
- Improvement at implementation level
  - Reduction btw WkGrp done once
  - More work in kernel
  - Hide the kernel latency

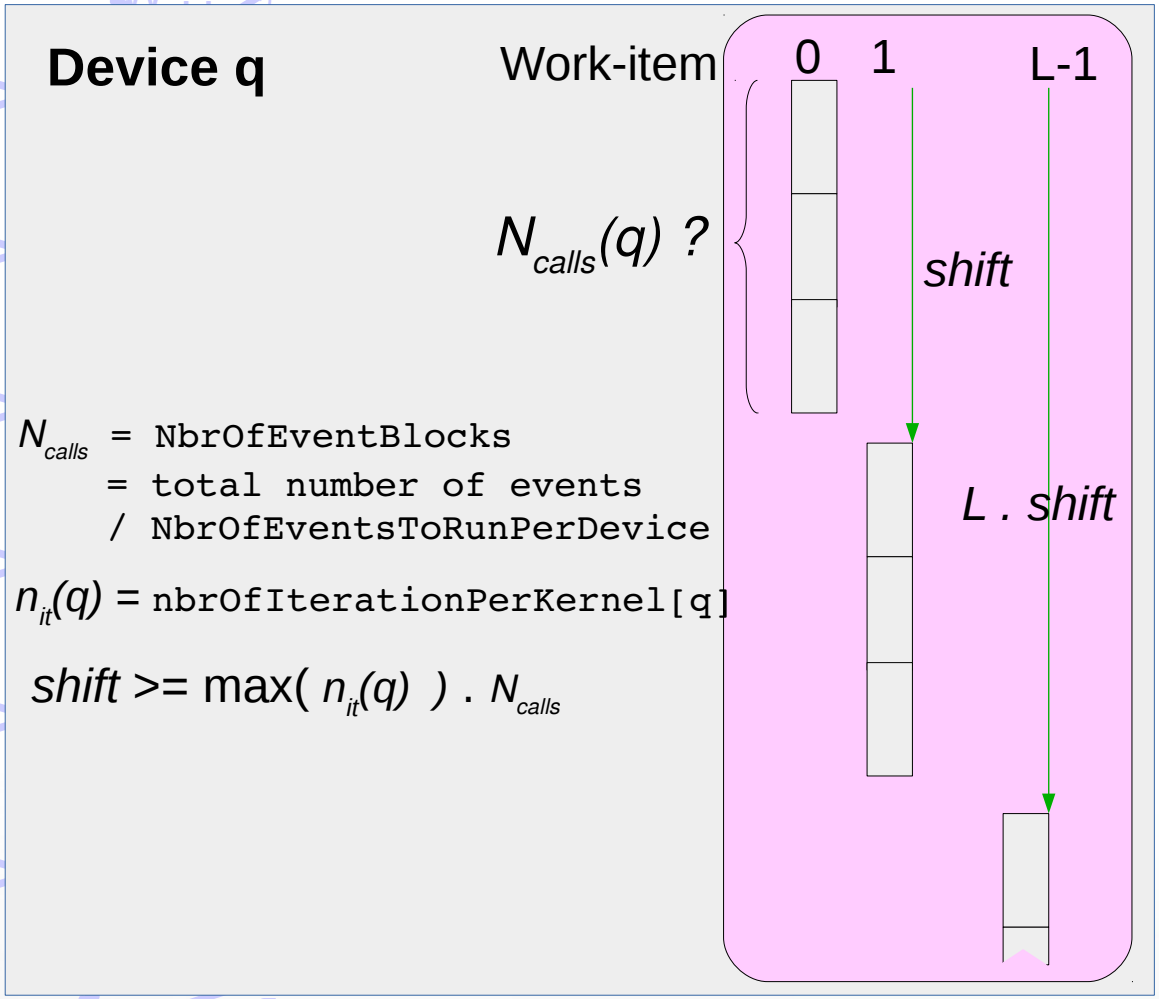
N_d = `NbrOfEventsToRunPerDevice`

$n_{it}(q)$ = `nbrOfIterationPerKernel[q]`

$n_{wg}(q)$ = `workGroupSize[q]`

$N_d = n_{it}(q) \cdot n_{wg}(q)$

MultiDevice RNG



- Result according to value and std error
- Not predictable order in work distribution (Device order not known)
- Re-find the initial value since using the same kind of hardware

MultiDevices
Device parallelism

```
void OCLManager::run( bool selectedQueues[] ,
                      cl_long nbrOfEvents ) {
 cl_long nbrSubmittedEvents=0, nbrEndedEvents=0;
 int q_max = queues_.size();
 // Main loop : until all events are done
 for (; ( nbrEndedEvents != nbrOfEvents); ) {
  for ( int q = 0; q < q_max; q++) {
   // Are Device task/ finished ?
   if( queueStatus_[q].status == ComputationCompleted ) {
    // Terminate computation
    queueStatus_[q].status  = ComputationNotActive;
    queues_[q].flush();
    // Update the number of proceeded events
    nbrEndedEvents += queueStatus_[q].nbrOfEvents;
    if ( nbrSubmittedEvents != nbrOfEvents ) {
     // Start a new computation
     cl_long nEv = min(nbrOfEvents - nbrSubmittedEvents,
                 (cl_long) NbrOfEventsToRunPerDevice );
     run( q, nEv );
     // Update the number of proceeded events
     nbrSubmittedEvents += nEv;
    }  }  }
    nanosleep( &delay, NULL);
  } }
```

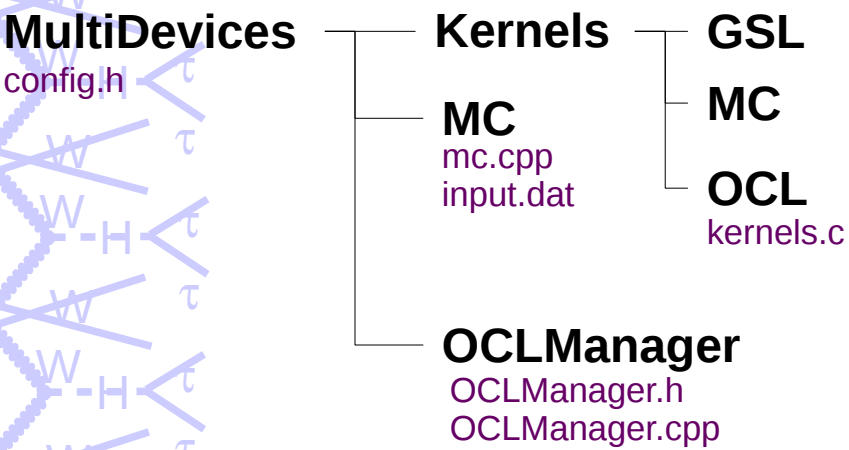
Simplified implementation of work distribution among the devices

- Main idea:

  Provide work to a device on demand
- Two embedded loops:
  - Event loop
  - Device loop
- If a device is "idle" then start a new computation
- `selectedQueues` for testing

Expected a good load-balancing between all the devices

MultiDevices OpenCL implementation



OneDevice → MultipleDevices

- kernels unchanged
- adding Kernels/OCL directory for API part
- Modification of Kernels/MC/stastistics.cpp for reduction part.

MultiDevices Hands on (2)

OneDevice

	C++ 1-core	OCL 16-cores	OCL K20	OCL Titan	OCL X. Phi	AMD FirePro
Time/ev (µs)	2.45	0.160	0.112	0.0870	0.157	
Speed-up	1	15.3	21.9	28.2	15.6	

MultiDevice
One device

	C++ 1-core	OCL 16-cores	OCL K20	OCL Titan	OCL X. Phi	AMD FirePro
Time/ev (µs)	2.45	0.0993	0.111	0.0917	0.149	
Speed-up	1	24.7	22.1	26.7	16.4	

MultiDevice
All devices

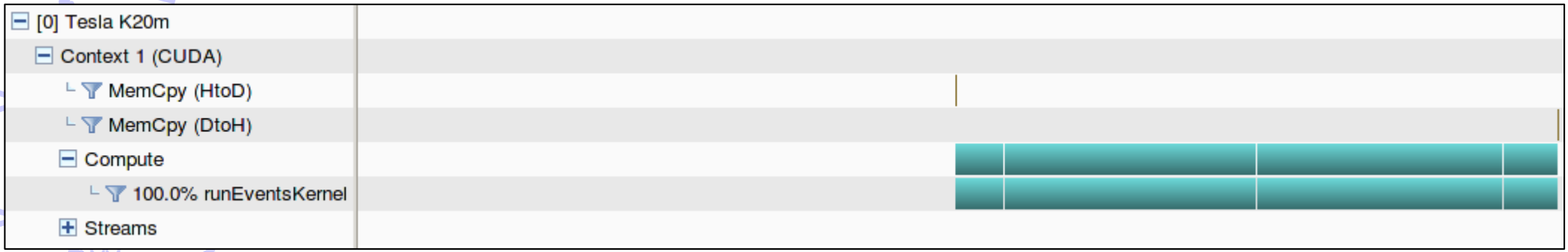
	C++ 1-core	OCL 16-cores	OCL K20	OCL Titan	OCL X. Phi	AMD FirePro
Time/ev (µs)	2.45		0.0389	0.0167	0.05169	
Speed-up	1		63.0	146.7	47.48	

MultiDevices
cl.hpp & NVProfiler (1)

`cl.hpp` developed @LLR

- Route OCL methods to CUDA calls
- Only switch the header cl.hpp
- Beta release
- Allow to use `nvprof`, `cuda-gdb`

MultiDevices
cl.hpp & NVProfiler (2)



Analysis | Details | Console | Settings

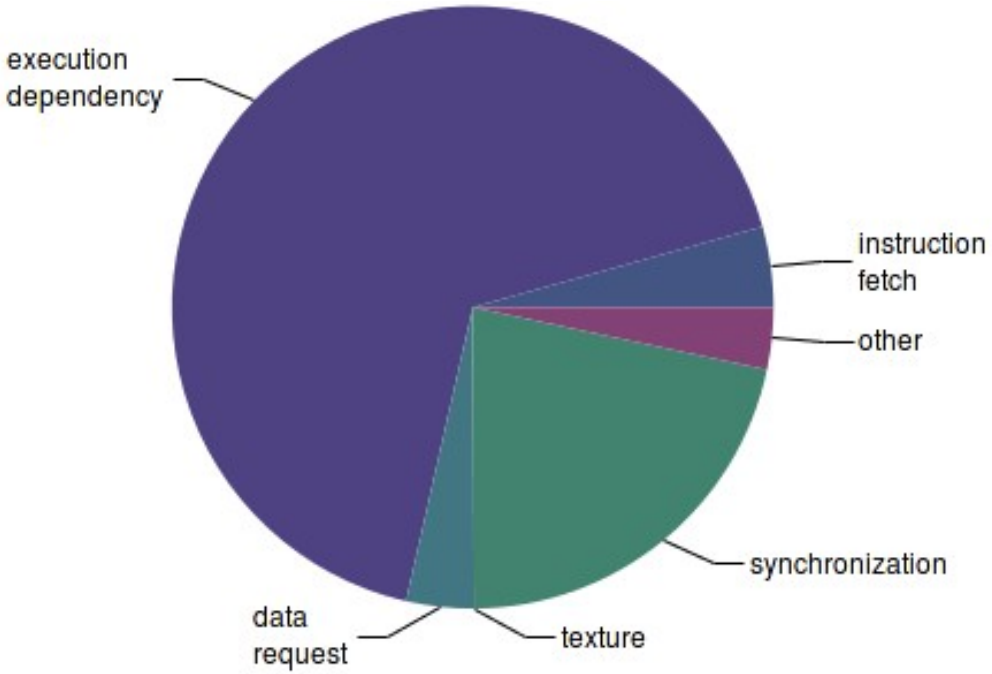
Name	Start Time	Duration	Grid Size	Block Size	Regs	Static SMem	Dynamic SMem	Size	Throughput
Memcpy HtoD [sync]	4.601 s	1.44 µs	n/a	n/a	n/a	n/a	n/a	320 bytes	211.93 MB/s
Memcpy HtoD [sync]	4.601 s	4.608 µs	n/a	n/a	n/a	n/a	n/a	20 KB	4.14 GB/s
Memcpy HtoD [sync]	4.601 s	992 ns	n/a	n/a	n/a	n/a	n/a	72 bytes	69.22 MB/s
Memcpy HtoD [sync]	4.602 s	1.405 ms	n/a	n/a	n/a	n/a	n/a	4.75 MB	3.3 GB/s
runEventsKernel	4.607 s	46.355 ms	[40,1,1]	[64,1,1]	84	0	0	n/a	n/a
runEventsKernel	4.654 s	46.295 ms	[40,1,1]	[64,1,1]	84	0	0	n/a	n/a
runEventsKernel	4.7 s	46.328 ms	[40,1,1]	[64,1,1]	84	0	0	n/a	n/a
runEventsKernel	4.746 s	46.323 ms	[40,1,1]	[64,1,1]	84	0	0	n/a	n/a

Device usage ok

nvprof warn: GlobalSize = n x 130 blocks = n x 8320

MultiDevices
cl.hpp & NVProfiler (3)

nvprof report



Stall Reasons

- Latency of arithmetic operation: OK
- Heavy synchronizations optimization → speed-up 34

MultiDevices
cl.hpp & NVProfiler (4)

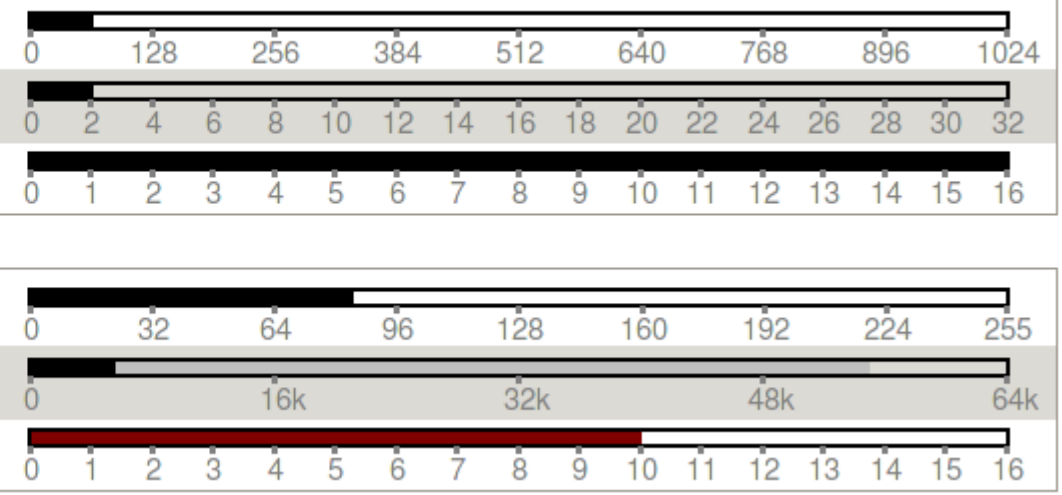
Warp & register usage

Warps

Threads/Block		64	1024
Warps/Block		2	32
Block Limit		16	16

Registers

Registers/Thread		84	255
Registers/Block		5632	65536
Block Limit		10	16



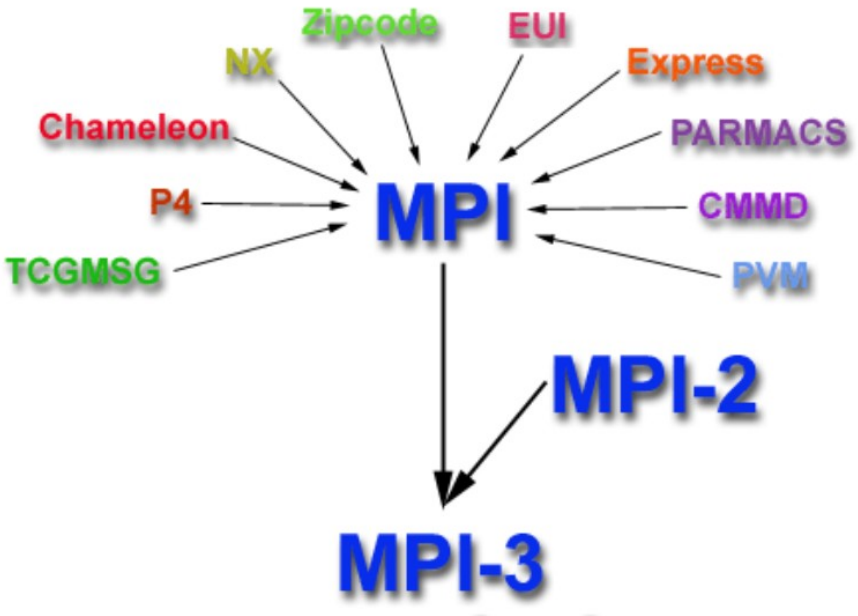
- Warp usage OK: application seems to take advantage of HW resources
- Register usage: 84 per threads → 84 x 64 x 10 = 53 760 reg. → limit the # of block (10) in //
- Potential optimization (<60%): to investigate

OpenCL & MPI

Part 4: OpenCL & MPI

- MPI standard
- Code $e^+e^- \rightarrow \gamma\gamma\gamma$

OpenCL & MPI
MPI Standard (1)

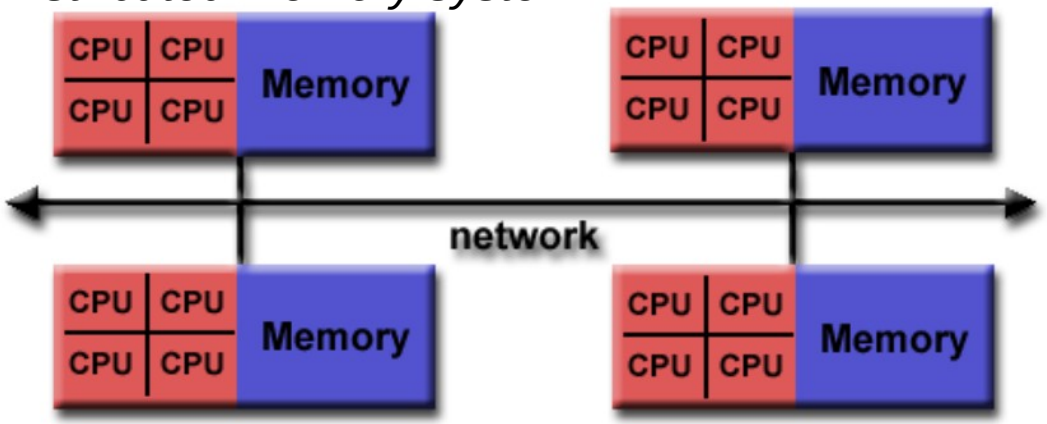


- MPI-1 the first MPI standard, called MPI-1 (1994). Success: well suited to take advantage of massively parallel super-computer
- In 1998 MPI-2 new standard (MPI-IO for // I/O, NASA)
- The most popular implementations:
  - MPICH: Argonne National Laboratory (ANL) and Mississippi State University, IBM
  - Open MPI: merging FT-MPI, LA-MPI, LAM/MPI, and PACX-MPI
- Supercomputer vendors implement their own version

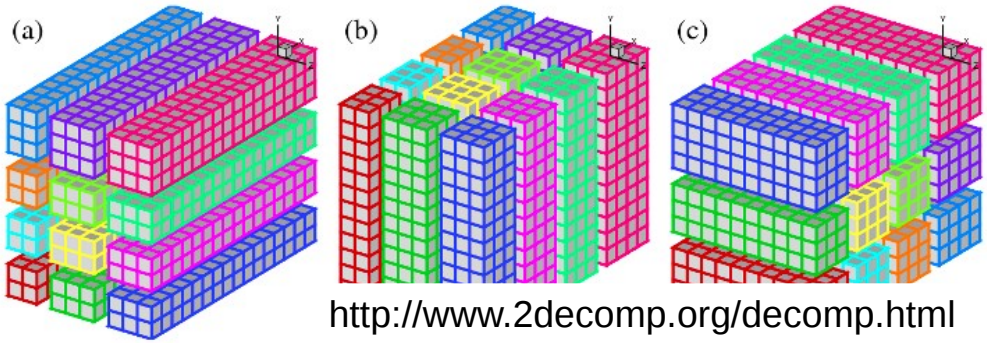
https://computing.llnl.gov/tutorials/mpi

OpenCL & MPI
MPI Standard (2)

Distributed memory system



Domain decomposition



http://www.2decomp.org/decomp.html

Numerous Large memory applications take advantage of Domain Decomposition

http://www.mcs.anl.gov/research/projects/mpi/tutorial

MPI function classification

Data locality & programming model

hardware platform dependent

- Shared memory system (OpenMP)
- Distributed memory system (MPI)

Types of Parallel Computing Models

application dependent

- Data Parallel - the same instructions are carried out simultaneously on multiple data items (SIMD). Independent tasks
- Task Parallel - different instructions on different data (MIMD). Concurrent tasks (DAG)
- SPMD (single program, multiple data) all processes have (almost) the same role (domain decomposition)
- MPMD (multiple program, multiple data) all processes have not the same role (Master/Slave model, …)

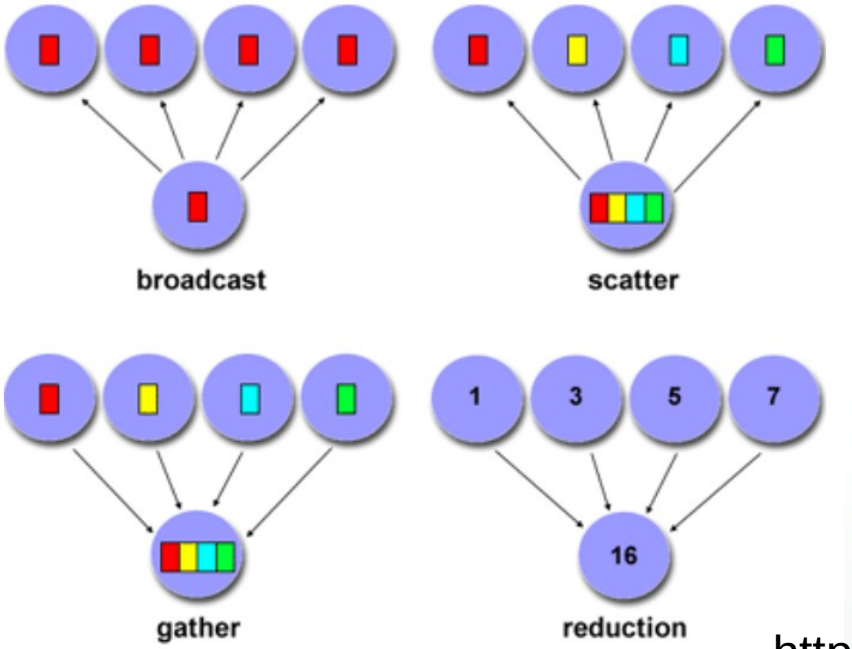
Hybrid programming (MPI+OpenMP)

OpenCL & MPI
MPI Standard (3)

Point to point communications

Process 0	Process i
send(i, data)	...
	receive(0, data)

Collective communications

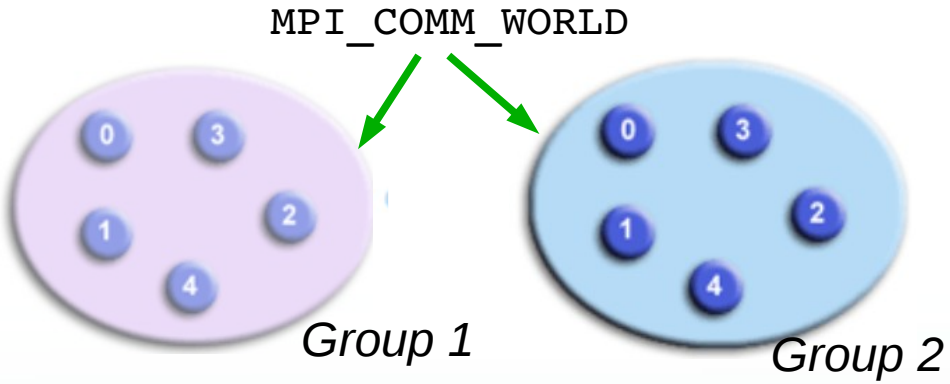


MPI function classification

One-sided communications

Process 0	Process i
put(i, data)	?
...	...
?	get(0, data)

Synchronization (barrier, communicator)



Blocking/non-blocking sub-classes

https://computing.llnl.gov/tutorials/mpi/

OpenCL & MPI
MPI Standard (4)

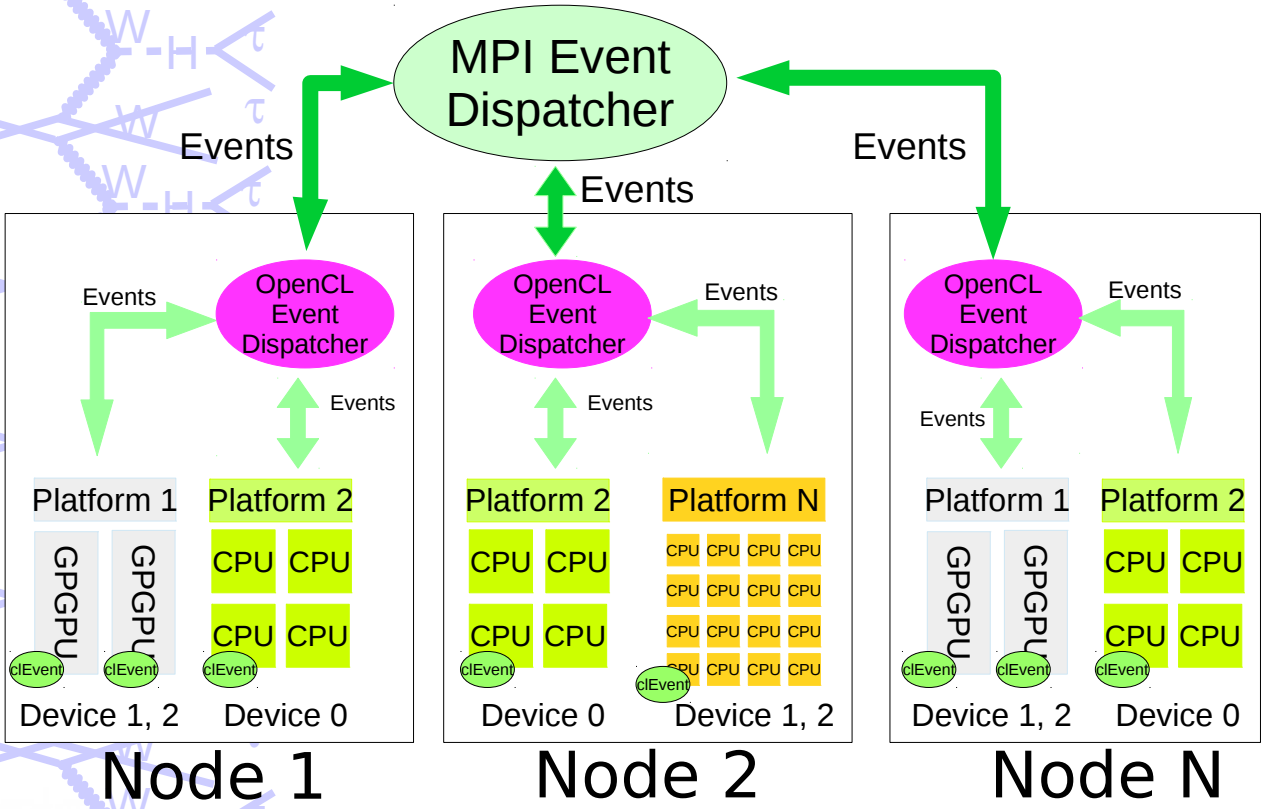
Simple example

```
# include "mpi.h"
# include <stdio.h>

main(int argc, char *argv[])  {
 int numtasks, rank, dst, src, rc, count, tag=1;
 char inmsg, outmsg='x';

 MPI_Status Stat;
 MPI_Init( &argc, &argv);
 MPI_Comm_size( MPI_COMM_WORLD, &numtasks);
 MPI_Comm_rank( MPI_COMM_WORLD, &rank);
 if (rank == 0) {
   dst = 1; src = 1;
   rc = MPI_Send(&outmsg, 1, MPI_CHAR, dst, tag, MPI_COMM_WORLD);
   rc = MPI_Recv(&inmsg , 1, MPI_CHAR, src, tag, MPI_COMM_WORLD, &Stat);
 } else if (rank == 1) {
   dest = 0; src = 0;
   rc = MPI_Recv(&inmsg , 1, MPI_CHAR, src, tag, MPI_COMM_WORLD, &Stat);
   rc = MPI_Send(&outmsg, 1, MPI_CHAR, dst, tag, MPI_COMM_WORLD);
 }
 rc = MPI_Get_count(&Stat, MPI_CHAR, &count);
 printf("Task %d: Received %d char(s) from task %d with tag %d \n",
        rank, count, Stat.MPI_SOURCE, Stat.MPI_TAG);
 MPI_Finalize();
}
```

OpenCL & MPI Code e+e- → γγγ (1)



- Work distribution like `MultiDevices` version
- Master/Slave model (load-balancing)
- *Same* message structure
- Restriction: no RNG shift

2 standards to take into account distributed nodes (MPI), cores(OCL), CPU vectorial HW, accelerator devices (OCL)

OpenCL & MPI
MPIDispatcher::run()

```
if ( processID_ == 0 )  {
  // Master
  // Master Event Loop
  for (msg_sent = 0;
                  newEventsRead || (msg_sent != msg_received); ) {
    // While no message in box
    for ( ; spin < spinWait_; spin++) {
      for ( msgInBox=1; msgInBox != 0 ; ) {
        // Test if message in the box
        MPI_Iprobe(  ..., resultTag_, … , &msgInBox, &status);
        if ( msgInBox != 0 ) {
          // A message is in the box, get the message
          MPI_Recv( eventList, msg_size, …,  resultTag_, ... , &status);
          // Is message non-empty message (start message)
          if ( eventList[0].MPIInfo_ != voidResult_ ) {
            // Store result ...
            msg_received++;
          }
          // Send a new work to do the same Worker
          if ( newEventsRead ) {
            MPI_Send ( eventList, ..., status.MPI_SOURCE, workTag_, ...);
            mpi_iter++; msg_sent++;
          }
        } // End a message in box
      } //
      nanosleep( &MPIdelay_, &effDelay_ );
    } // Spin loop
  } // Loop on events

  // Send terminate message to Workers
  for ( p=1; p < nbrOfProcess_; p++ ) {
    eventList[0].MPIInfo_ = stopWork_;
    MPI_Send (eventList, msg_size, MPI_BYTE, p, workTag_, ...);
}
```

```
  ...
} else {
  //
  //  Workers
  int msg_received = 0;
  //
  //  Init
  //  Send void result message to start the Master
  eventList[0].MPIInfo_ = voidResult_;
  MPI_Send ( eventList, msg_size, … , 0, resultTag_, … );
  // Waiting work messages until terminate message
  msg_received = 0;
  int continueWork = 1;
  for ( ; continueWork == 1 ; ) {
    // Received work message
    MPI_Recv( eventList, … ,, 0, workTag_, … , &status);
    // Is a terminate message
    if ( eventList[0].MPIInfo_ == stopWork_ ) {
      // Stop the worker
      continueWork = 0;
    } else {
      // Worker compute : run Device Dispatcher
      OCLMgr_->run( selectedQueues, run->nbrOfEvents );
      // Send back result
      eventList[0].MPIInfo_ = processID_;
      MPI_Send (eventList, msg_size, ..., 0, resultTag_, … );
      // Next event buffer
      msg_received++;
    }
  }  // End message loop
}
```

OpenCL & MPI Execution

- Bitbucket oclmpiref/OCL-MPI
- Batch example
- MPI environment

```
$ . /usr/share/Modules/init/bash
$ unset MODULEPATH
$ module use /opt/exp_soft/vo.llr.in2p3.fr/modulefiles
$ module load mpi/openmpi-bash-4.1$
```

- `qsub -q k20 job.sh`
- For more:
  https://llrgit.in2p3.fr/GridCL/GridCL/wikis/Batch

OpenCL & MPI

Conclusion

- Sum-up: HPC & Accelerators
- Other experiments
- Perspectives

Conclusion
HPC & Accelerators

- HPC context to exploit CPU time consuming
- Accelerator HW well suited to challenge the "Power and Energy Wall " … no other way, FGGA ?
- OpenCL is designed to HPC application (vectorized and //ize inside one node

OpenCL

- API part not easy the 1st implementation
- Designed to handle simultaneously heterogeneous HW
- Lake of tools (Performance, debugging) … Intel, AMD
- Libraries
  https://www.khronos.org/opencl/resources
  Libraries and Frameworks with OpenCL

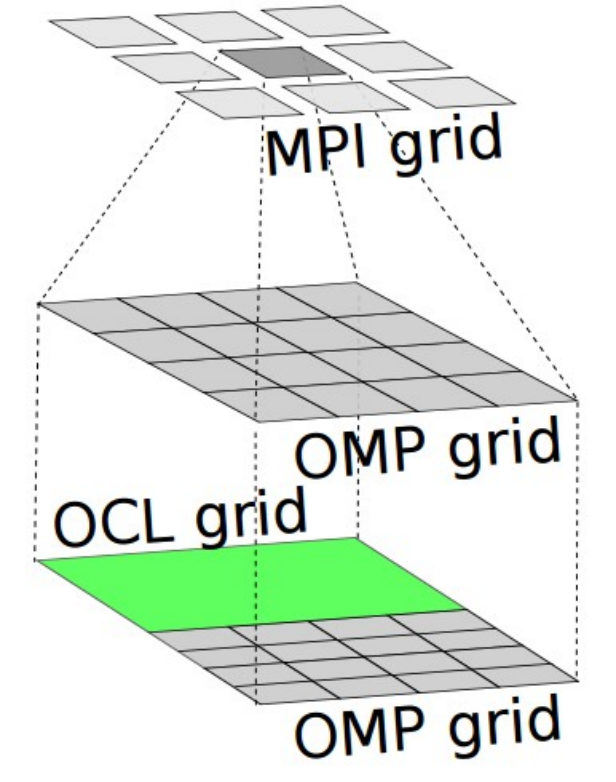
OpenMP 4.x

- Easy to develop
- But …More & more unreadable code
- Not mature compilers
- Handling heterogeneous hardware ?
- Free (cheap) implementations ...
- … keep in touch with OMP

Conclusion
LLR experiences

Hydro (JDEV 2013)

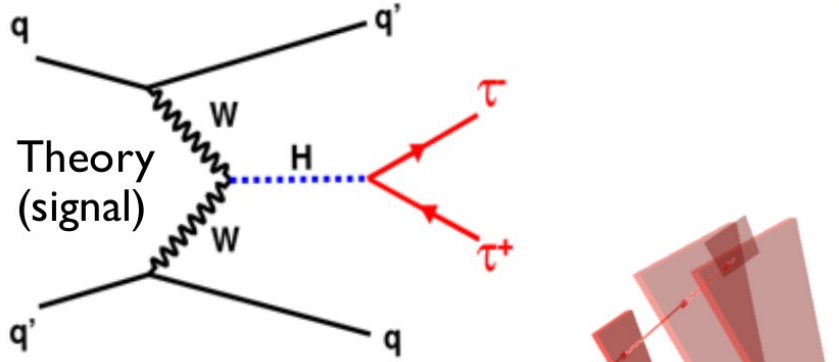
- Multi-level Domain decomposition

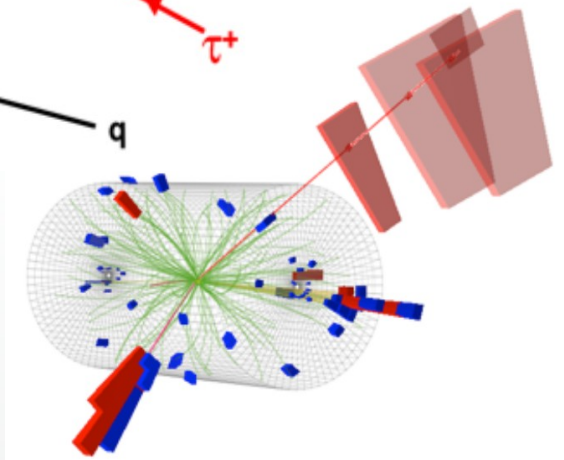


ME Method (CMS)

- Data // (MPI level)
- //zing the integration computation (OCL level)

$$p_{hyp}(\boldsymbol{y}) = \frac{1}{\sigma_{tot}} \sum \int d\boldsymbol{x}\, |M_{\omega}(\boldsymbol{x})|^2 W(\boldsymbol{x}|\boldsymbol{y})$$





Conclusion Perspectives

To investigate

- Need a high level description to harness changing technologies
- Code generator, mini-DSL, ...
- Or class abstraction OCCA, kokkos, ...

Talk of Andreas

- Modern tool to tackle evolving technologies
- Description at high level