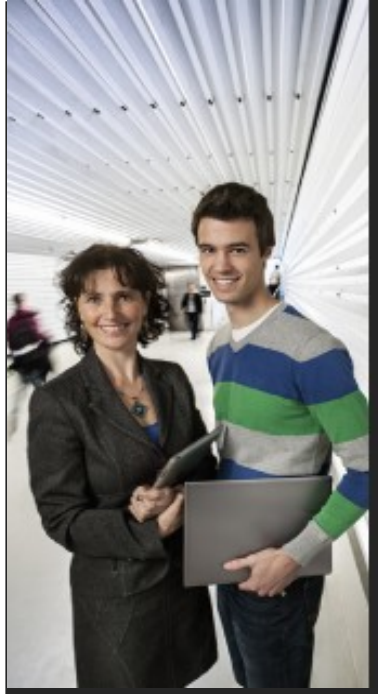
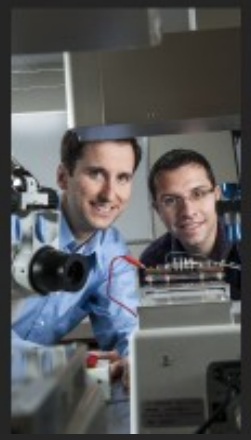
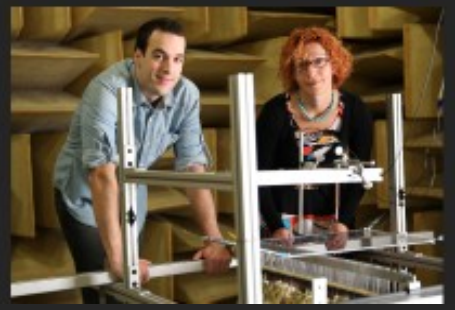


POLYTECHNIQUE MONTRÉAL

WORLD-CLASS ENGINEERING

LTTng & GPU Tracing

Presented by : David Couturier

Updated December2014

CONTENTS

What are GPUs

- Graphical Processing Unit
- Graphical acceleration of multimedia (games, movies, ...)
- Co-processor for computations (GPGPU: General Purpose Graphical Processing Unit)

Graphical Acceleration
Targeted library: OpenGL
- Video games
- Video Playback

GPGPU
Targeted library: OpenCL
- Scientific calculations
- Application parallel computing acceleration

Graphical Accelerators (challenges)

- Slow video game
- “Choppy” display rate
- Frame “miss”
- Guaranteed minimum fps (military use)

Graphical Accelerators (available tools)

- AMD GPUPerfStudio
- NVIDIA Nsight
- API Trace
- VoGL (Valve's OpenGL profiling tool)
- GPU PerfStudio (AMD)
- GDEBugger
- [...]

GPGPUs (challenges)

- Maximize resource utilization
  - Multiple SIMD
  - Thread Pool
  - Multiple Context (multiple programs running concurrently)
- Minimize idle time due to data transfers

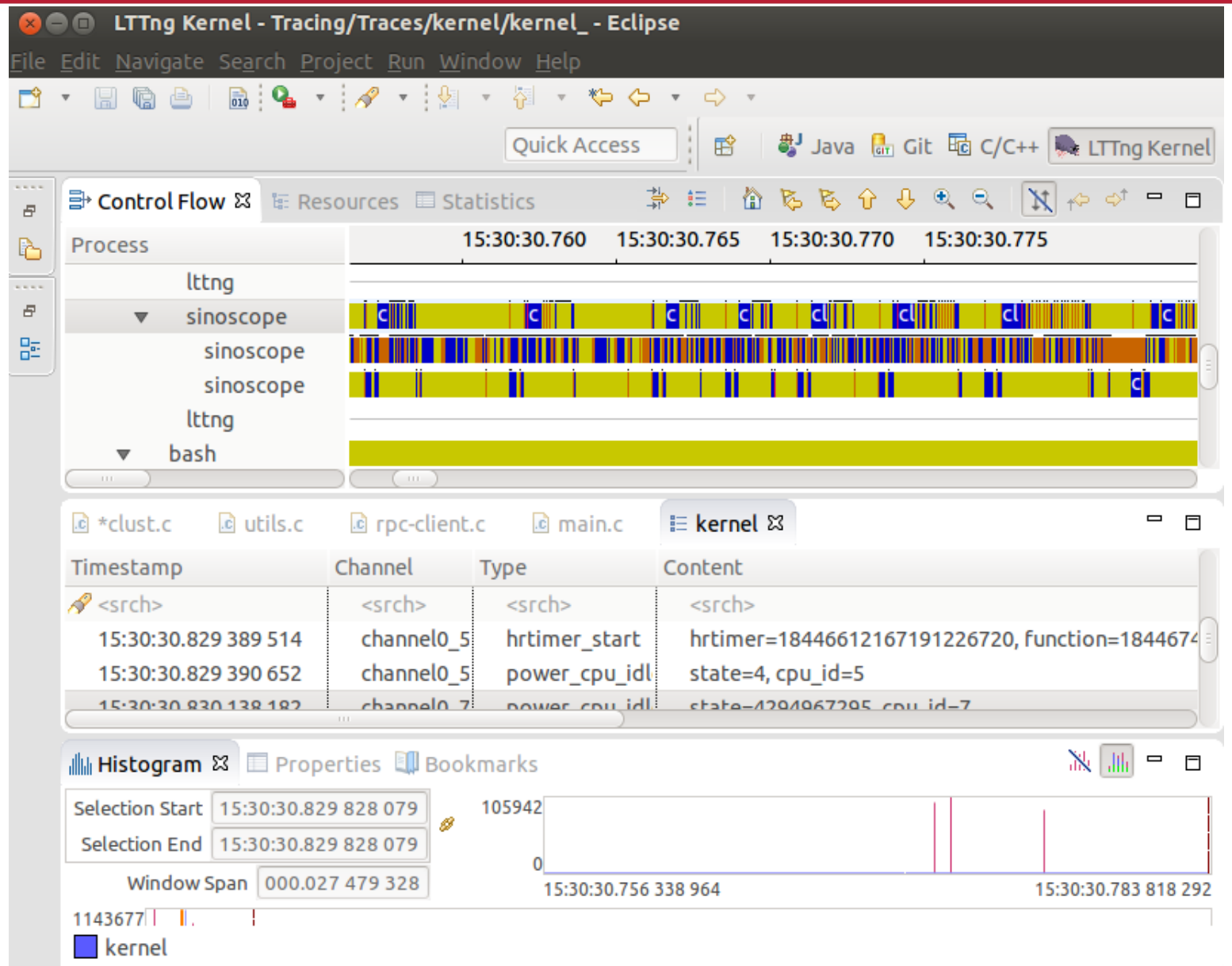
GPGPUs (available tools)

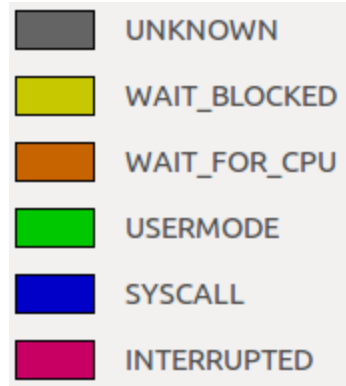
- Propriatery only!
  - AMD's CodeXL
  - NVIDIA's Nsight

AMD CODEXL

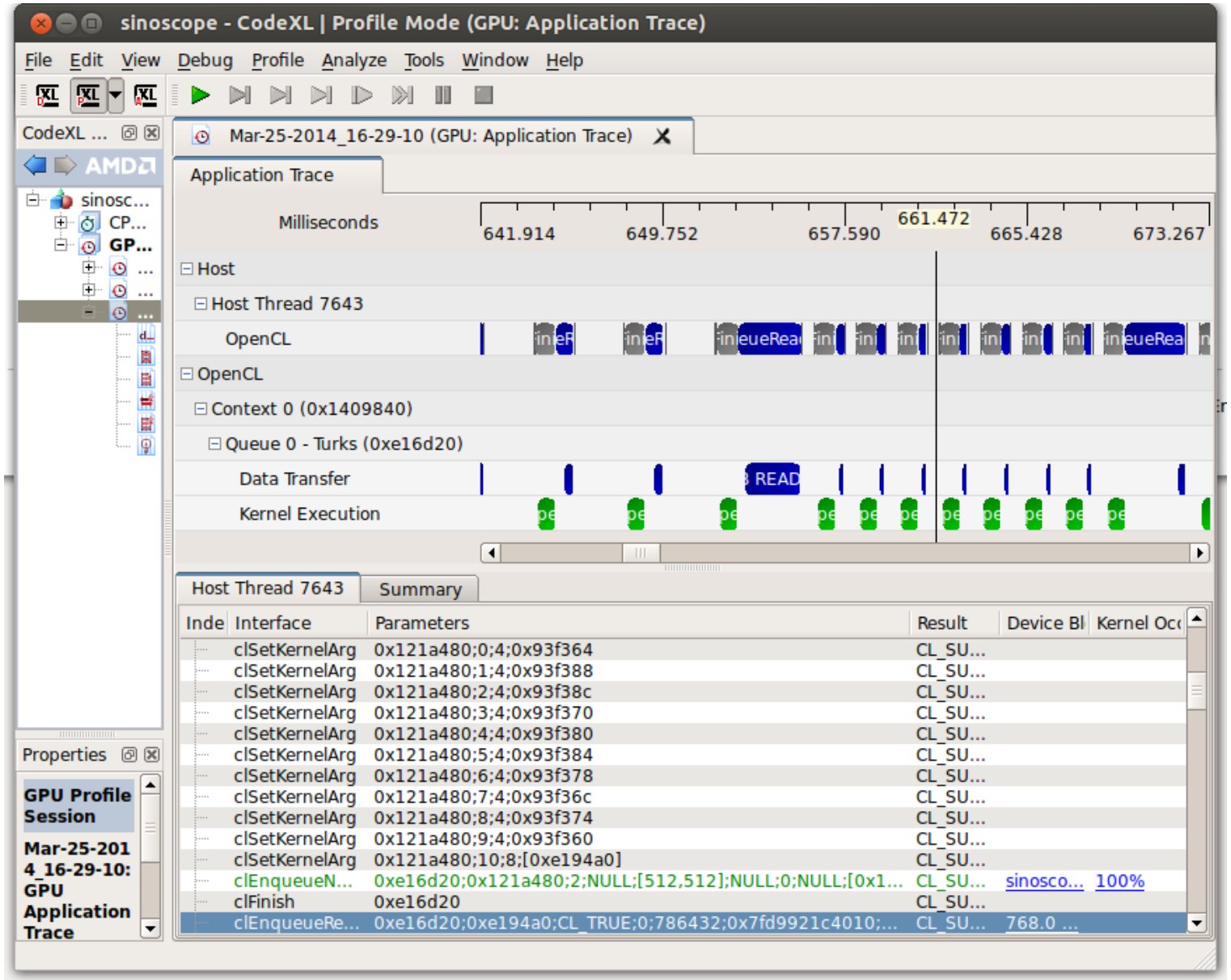
NVIDIA® Nsight™

Current Tracing capabilities





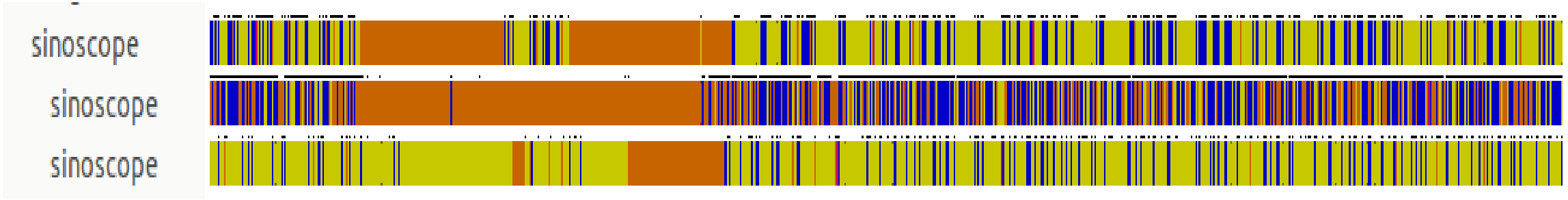
GPGPUs (available tools)



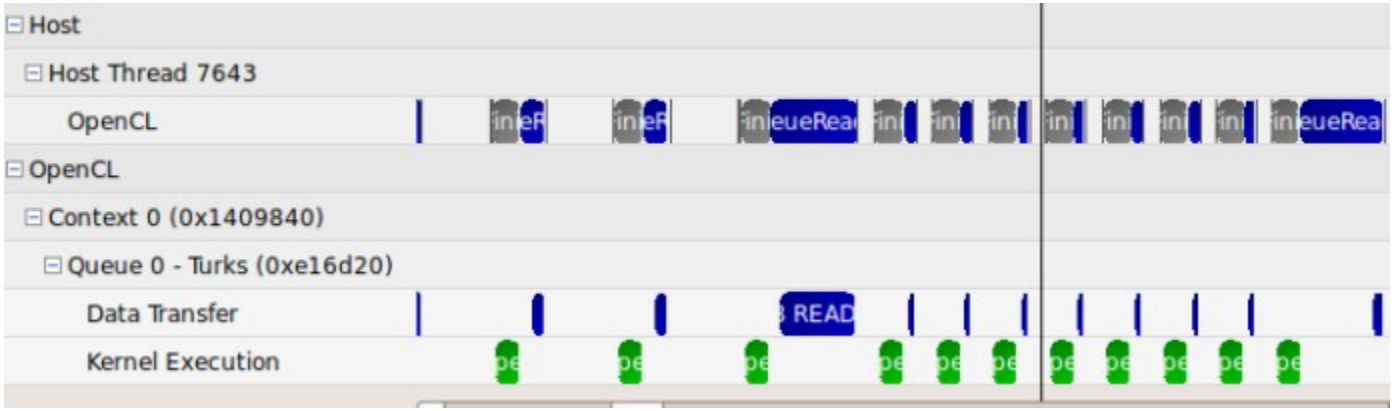
Inde	Interface	Parameters	Result	Device Bl	Kernel Occ
	clSetKernelArg	0x121a480;0;4;0x93f364	CL_SU...		
	clSetKernelArg	0x121a480;1;4;0x93f388	CL_SU...		
	clSetKernelArg	0x121a480;2;4;0x93f38c	CL_SU...		
	clSetKernelArg	0x121a480;3;4;0x93f370	CL_SU...		
	clSetKernelArg	0x121a480;4;4;0x93f380	CL_SU...		
	clSetKernelArg	0x121a480;5;4;0x93f384	CL_SU...		
	clSetKernelArg	0x121a480;6;4;0x93f378	CL_SU...		
	clSetKernelArg	0x121a480;7;4;0x93f36c	CL_SU...		
	clSetKernelArg	0x121a480;8;4;0x93f374	CL_SU...		
	clSetKernelArg	0x121a480;9;4;0x93f360	CL_SU...		
	clSetKernelArg	0x121a480;10;8;[0xe194a0]	CL_SU...		
	clEnqueueN...	0xe16d20;0x121a480;2;NULL;[512,512];NULL;0;NULL;[0x1...	CL_SU...	sinosco...	100%
	clFinish	0xe16d20	CL_SU...		
	clEnqueueRe...	0xe16d20;0xe194a0;CL_TRUE;0;786432;0x7fd9921c4010;...	CL_SU...	768.0 ...	

Separate Traces!?!

LTTng Kernel Trace



AMD's CodeXL Trace



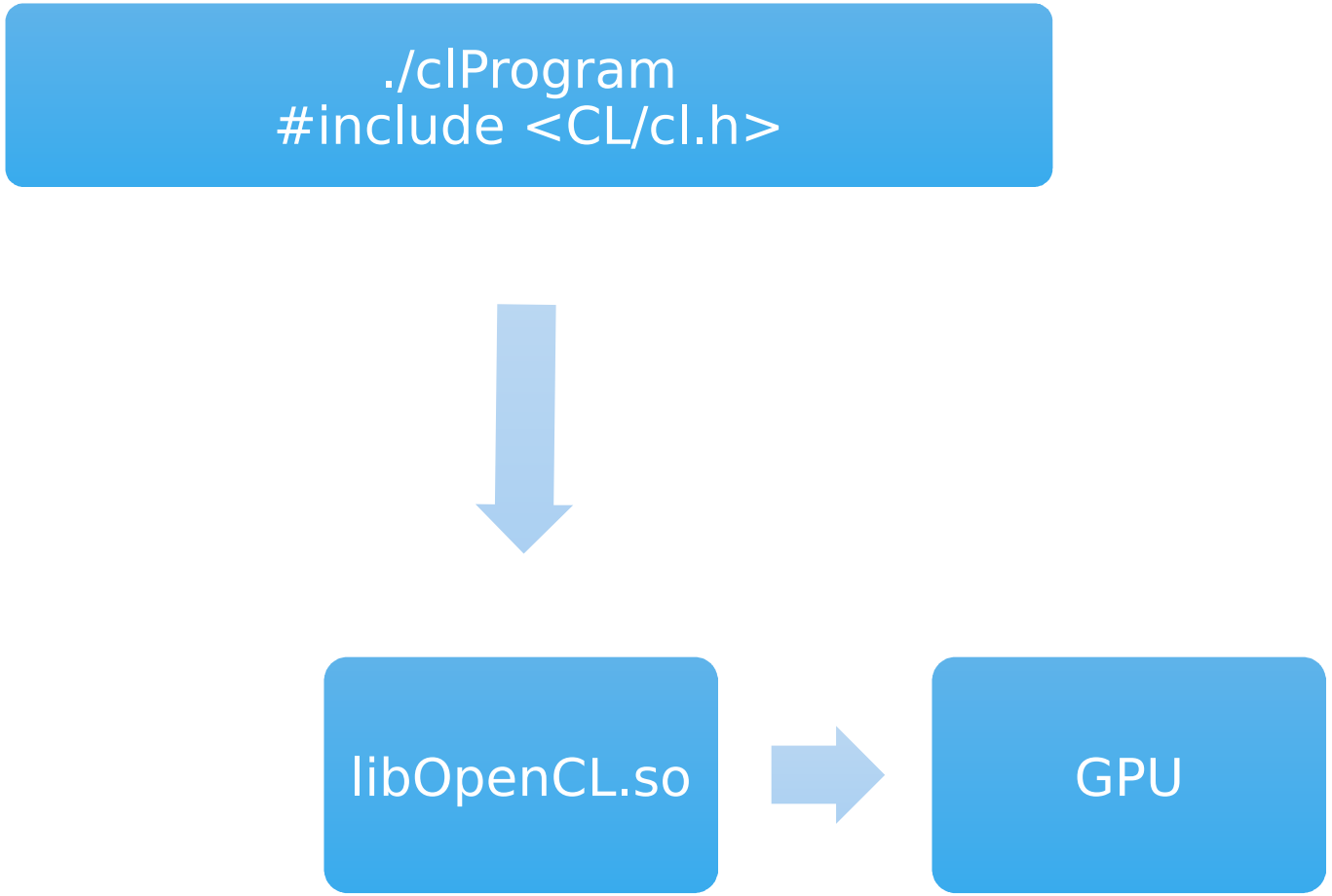
Library Overrider

- Java program
- Automatic wrapping & instrumentation of the OpenCL library

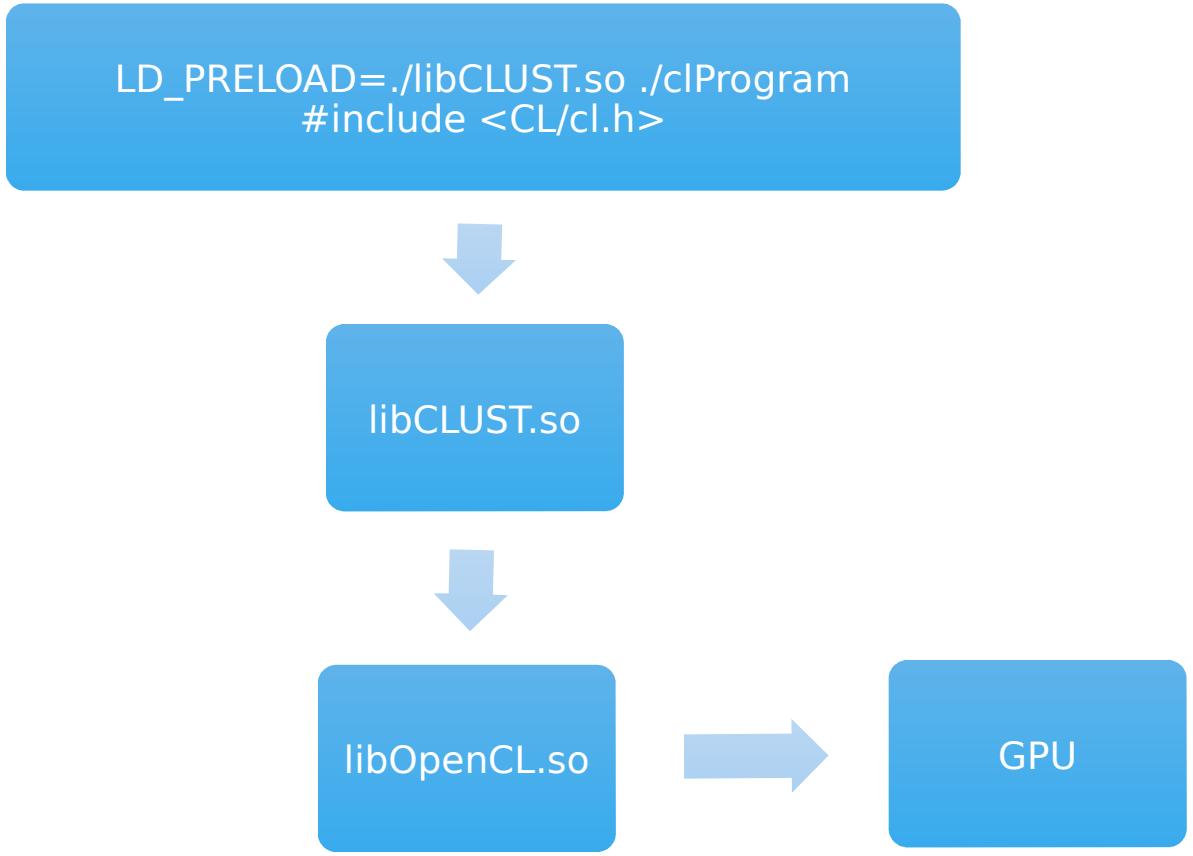
CLUST

- Open Source wrapper of OpenCL
- Enables tracing any OpenCL application
- Simple to trace
- Records in CTF (Common Trace Format)
    - High performance tracing format
- Using LTTng UST tracepoints
- Performance target: < 5% slowdown on the system*
- Allows to trace Kernel Events at the same time, on the same trace.

Normal Flow



CLUST Flow


LD_PRELOAD=./libCLUST.so ./clProgram
#include <CL/cl.h>
libCLUST.so
libOpenCL.so
GPU

Library Constructor

```
__attribute__((constructor)) void libCLUST() {
        void* libcl_ptr;
        dlerror();
        libcl_ptr = dlopen(LIBCL_NAME, RTLD_LAZY);

        if(!libcl_ptr) {
                fprintf(stderr, "%s: Unable to load %s\n", LIB_NAME, LIBCL_NAME);
                exit(EXIT_FAILURE);
        }

        dlerror();
        reallib_clGetPlatformIDs = (cl_api_call_clGetPlatformIDs) dlSymFunction(li
```

dlSym for all the OpenCL functions

```
dlerror();
reallib_clGetPlatformIDs = (cl_api_call_clGetPlatformIDs) dlSymFunction(libcl_ptr, "clGetPlatformIDs");
reallib_clGetPlatformInfo = (cl_api_call_clGetPlatformInfo) dlSymFunction(libcl_ptr, "clGetPlatformInfo");
reallib_clGetDeviceIDs = (cl_api_call_clGetDeviceIDs) dlSymFunction(libcl_ptr, "clGetDeviceIDs");
reallib_clGetDeviceInfo = (cl_api_call_clGetDeviceInfo) dlSymFunction(libcl_ptr, "clGetDeviceInfo");
reallib_clCreateContext = (cl_api_call_clCreateContext) dlSymFunction(libcl_ptr, "clCreateContext");
reallib_clCreateContextFromType = (cl_api_call_clCreateContextFromType) dlSymFunction(libcl_ptr, "clCreateContextFromType
reallib_clRetainContext = (cl_api_call_clRetainContext) dlSymFunction(libcl_ptr, "clRetainContext");
reallib_clReleaseContext = (cl_api_call_clReleaseContext) dlSymFunction(libcl_ptr, "clReleaseContext");
reallib_clGetContextInfo = (cl_api_call_clGetContextInfo) dlSymFunction(libcl_ptr, "clGetContextInfo");
reallib_clCreateCommandQueue = (cl_api_call_clCreateCommandQueue) dlSymFunction(libcl_ptr, "clCreateCommandQueue");
reallib_clRetainCommandQueue = (cl_api_call_clRetainCommandQueue) dlSymFunction(libcl_ptr, "clRetainCommandQueue");
reallib_clReleaseCommandQueue = (cl_api_call_clReleaseCommandQueue) dlSymFunction(libcl_ptr, "clReleaseCommandQueue");
reallib_clGetCommandQueueInfo = (cl_api_call_clGetCommandQueueInfo) dlSymFunction(libcl_ptr, "clGetCommandQueueInfo");
reallib_clCreateBuffer = (cl_api_call_clCreateBuffer) dlSymFunction(libcl_ptr, "clCreateBuffer");
reallib_clCreateSubBuffer = (cl_api_call_clCreateSubBuffer) dlSymFunction(libcl_ptr, "clCreateSubBuffer");
reallib_clCreateImage2D = (cl_api_call_clCreateImage2D) dlSymFunction(libcl_ptr, "clCreateImage2D");
reallib_clCreateImage3D = (cl_api_call_clCreateImage3D) dlSymFunction(libcl_ptr, "clCreateImage3D");
reallib_clRetainMemObject = (cl_api_call_clRetainMemObject) dlSymFunction(libcl_ptr, "clRetainMemObject");
reallib_clReleaseMemObject = (cl_api_call_clReleaseMemObject) dlSymFunction(libcl_ptr, "clReleaseMemObject");
reallib_clGetSupportedImageFormats = (cl_api_call_clGetSupportedImageFormats) dlSymFunction(libcl_ptr, "clGetSupportedIma
reallib_clGetMemObjectInfo = (cl_api_call_clGetMemObjectInfo) dlSymFunction(libcl_ptr, "clGetMemObjectInfo");
```

Instrumentation of each OpenCL functions

```
cl_int clGetPlatformIDs(cl_uint num_entries, cl_platform_id * platforms,
        |       cl_uint * num_platforms) {
        // Trace API call begin
        tracepoint(clust_provider, clust_clGetPlatformIDs, num_entries,
        |       platforms, num_platforms, true);
        // Call to the real OpenCL library
        cl_int ret = reallib_clGetPlatformIDs(num_entries, platforms, num_platforms);
        // Trace API call end
        tracepoint(clust_provider, clust_clGetPlatformIDs, num_entries,
        |       platforms, num_platforms, false);
        return ret;
}
```

Instrumentation of device dependent API calls

```
cl_int clEnqueueNDRangeKernel(cl_command_queue command_queue, cl_kernel kernel, cl_uint work_dim,
        |       const size_t * global_work_offset, const size_t * global_work_size,
        |       const size_t * local_work_size, cl_uint num_events_in_wait_list,
        |       const cl_event * event_wait_list, cl_event * event)  {
        // Trace API Call begin
        tracepoint(clust_provider, clust_clEnqueueNDRangeKernel, command_queue, [...params...], event, true);
        if(event == NULL) { // If the given event is NULL, allocate and mark freeEvent = true
        |       event = (cl_event *)malloc(sizeof(cl_event));
        |       freeEvent = true;
        }
        // Call to the real OpenCL library
        cl_int ret = reallib_clEnqueueNDRangeKernel(command_queue, [...params...], event);
        reallib_clSetEventCallback(event, CL_COMPLETE, &clustEventCallback, freeEvent);
        // Trace API Call end
        tracepoint(clust_provider, clust_clEnqueueNDRangeKernel, command_queue, [...params...], event, false);
        return ret;
}
```

Recording device event info

Event Callback Function:

```
// Get event start time
reallib_clGetEventProfilingInfo(event, CL_PROFILING_COMMAND_START, sizeof(cl_ulong), &start, NULL);
// Get event end time
reallib_clGetEventProfilingInfo(event, CL_PROFILING_COMMAND_END, sizeof(cl_ulong), &end, NULL);
// Get event command name (CL_COMMAND_NDRANGE_KERNEL, CL_COMMAND_WRITE_BUFFER, ...)
reallib_clGetEventInfo(event,CL_EVENT_COMMAND_TYPE,sizeof(cl_command_type), &command, NULL);
// Get event queue id
reallib_clGetEventInfo(event,CL_EVENT_COMMAND_QUEUE,sizeof(cl_command_queue), &queue, NULL);
// Record with UST tracepoint
tracepoint(clust_provider, clust_device_event, queue, command, start, end);
if(freeEvent) {
        reallib_clReleaseEvent(event);
}
```

Matching trace’s time domains

System monotonic clock (for the API calls)

vs

Device’s clock

Impact on the program performance

- No significant data yet…
- Cost of running the library with the wrapper (no tracing) VS cost of running and tracing the program with the wrapper

- Calculate the cost of running the program with and without GPU tracing for a fixed number of iterations

and/or

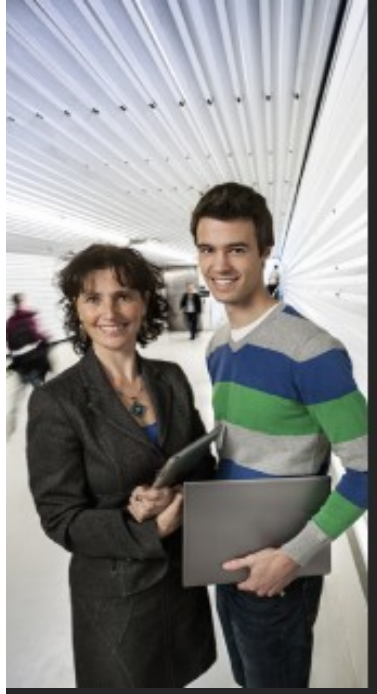
- Synthetic benchmark made specifically to measure the overhead on each functions

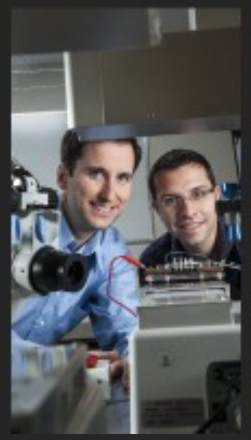
Future work: GLUST

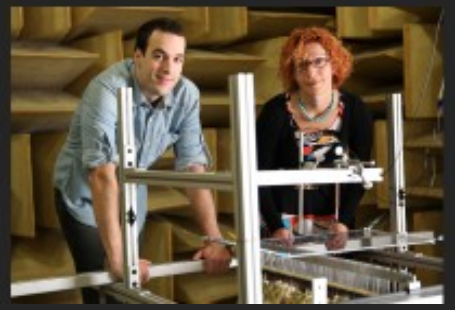
- Follow up of CLUST
- Same technique
- Possible collaboration with Samsung

CONCLUSION

- Open Source tracing tool for GPUs
- With performance as a priority
- Combine both Kernel trace and GPU trace



POLYTECHNIQUE MONTRÉAL

WORLD-CLASS ENGINEERING

THANK YOU!
Questions?