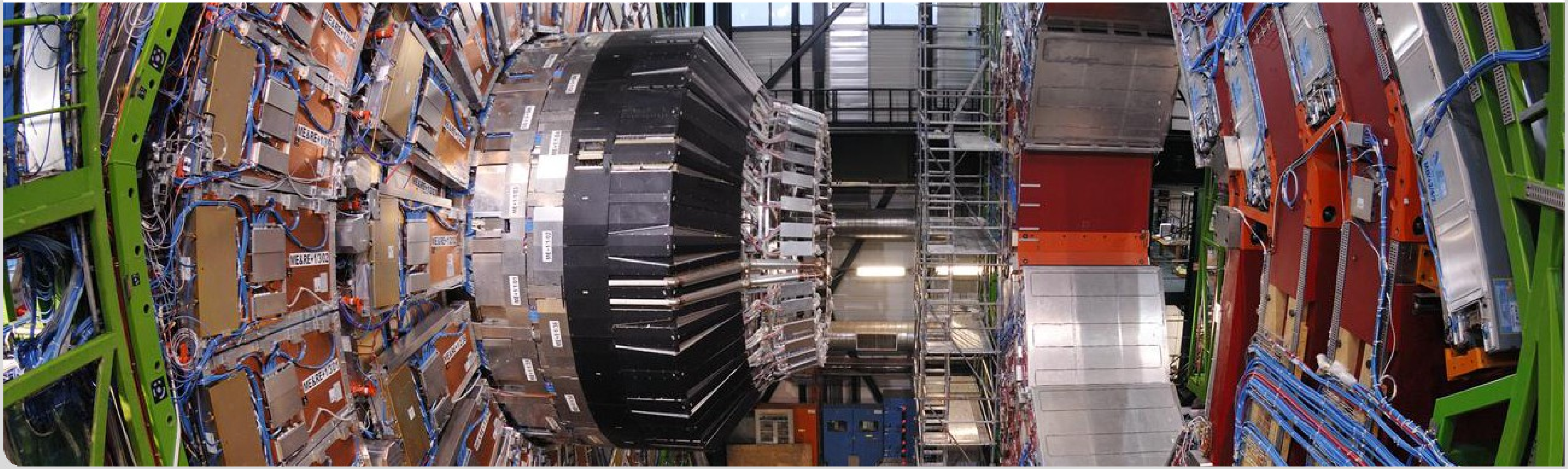
Research Project: OpenCL for Physics Applications

Thomas Hauth (CERN/KIT), Vincenzo Innocente (CERN),
Danilo Piparo (CERN), Benedikt Hegner (CERN)



Motivation

Goal:

Find a programming model to exploit current and future hardware for compute intensive tasks

Constraints:

- *Simplify multi-core programming, not complicate it*
- *Keep the code portable and as high-level as possible*
- *Don't program for specific hardware or instruction sets*

OpenCL (Open Computing Language)

- Standardized Framework for parallel programming on heterogeneous systems
- Managed by the Khronos Group (OpenGL, WebGL)
- Offers a common interface to run compute intensive tasks (so called Kernels) on CPU, GPU or other compute devices
- A subset of the C language is used to write these Kernels
- Platform implementations provided by hardware vendors

CPU GPU GPU CPU & GPU GPU . . .

High Portability

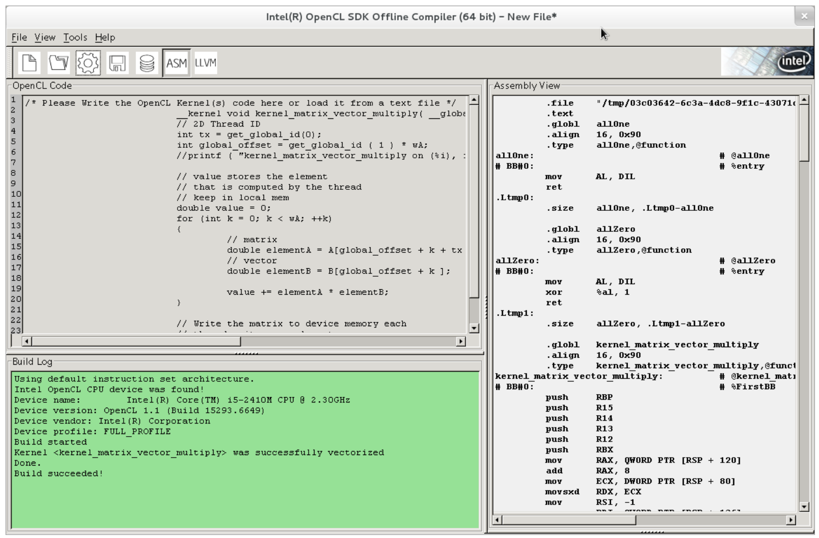
- The same Kernel and host code can be used to run on a multitude of compute hardware
- Platform implementations compile the Kernel's code to the machine instructions their hardware supports
  - Some tweaking can be done to better fit different memory layout and processor capabilities

http://www.khronos.org/opencl/

Our Choice: Intel's OpenCL SDK [1]

Reasons:

- Implements the OpenCL 1.1 specification for x86_64 CPUs >runs on our current hardware
- Available for free for the Windows and Linux platforms
- Automatically generates binaries which use the vector units of the CPU [2]
- Dispatches the Kernels to all available cores, no explicit multi-threading necessary



- *Intel Offline Compiler* allows to compile the Kernel code, look at the generated Assembly output and reports if the Kernel was successfully vectorized
- More tools available[3]: *Graphics Performance Analyzer* v 4.0 for OpenCL, *Amplifier XE Analysis* ..

[1] http://software.intel.com/en-us/articles/vcsource-tools-opencl-sdk/
[2] http://www.llvm.org/devmtg/2011-11/ - third talk -
[3] http://software.intel.com/en-us/articles/introduction-to-intel-opencl-tools/

Prototype: Simplified OpenCL Programming

- Using the OpenCL interface in C/C++ is cumbersome and error prone as a lot of code is necessary
- Memory allocation has to be done explicitly and cannot be done inside a kernel
- Kernels are only compiled during runtime, errors sometimes hard to detect

To decrease this overhead, we developed an OpenCL wrapper for C++

- Based on the OpenCLAM[1] project, but customized and extended
- Easy setup of the OpenCL runtime (~ 50 lines of C code to 2 lines C++)
- Kernels and their parameters can be easily defined and are checked for correctness by the host's C++ compiler (GCC in our case)
- Simplified memory management of buffer objects on the compute device
- Complete code example can be found in the Backup

Use Case: Matrix Algebra

- Data types for Matrices and Vectors have been implemented
- Math library to operate on Matrix and Vector data types (see Backup for example)
  - Matrix x Vector, Matrix x Matrix etc.

[1] http://code.google.com/p/openclam/

Summary & Outlook

- OpenCL is the emerging Industry Standard for portable high-performance computing
  - CPU, GPU portability
- With smart wrapper classes, OpenCL kernels are seamlessly integrated in the C++ code
  - It Works !
- Easy Kernel development by syntax and type check during compile time
- It was shown that common data types (Matrix, Vector) and mathematical operations can be provided to the user via high-level C++
- OpenCL on algorithm level is orthogonal to high-level module paralellism (see Chris Jones' talk)

Next Steps

- Implement selected parts of the CMS reconstruction outside the CMS software FW as OpenCL Kernels to quantify the possible gains
- Assess the portability of the current setup when running on GPU
  - See how much of the porting can already be handled inside the C++ wrapper (and therefore be kept away from the Kernel programmer)
- Test the multi-core scalability of Intel's OpenCL SDK with many cores (> 4)
- Quantitative comparisons to SMatrix performance hopefully to come next time

BACKUP

Prototype: Simplified OpenCL Programming

Complete Code Example

```
openclam::opencl wrapper;
openclam::context context( wrapper );

// define Matrix of size 10x10
typedef openclam::matrix<double,10> Matrix;

// initialize Matrix
std::vector < double >  arr(Matrix::value_elements, 1.0);
Matrix m1 ( arr, 1, wrapper, context );

double d2 = 23.0f;

// define kernel with all needed parameters
KERNEL2_CLASS( add_val , cl_mem, double  ,
               __kernel void add_val( __global double * a, const double b )
              {
                        a[ get_global_id( 0 ) ] += b;
              }) ( context );

// run kernel, with 2 parameters
add_val.run( m1.range_linear(), m1, d2 );

// get result
m1.to_array( arr, wrapper, context );
```

Prototype: Simplifying Matrix Operations

- ROOT's Similarity Operation is heavily used in the CMS Track Reconstruction Kalman Filter:

$$B = U * A * U^T$$

- ROOT's call instruction (for one track candidate)

```
err_new = ROOT::Math::Similarity( prediction_matrix, err_matrix );
```

- Starting OpenCL Kernels to do the same task in our prototype:
  - A buffer to hold temporary values during the calculation (`track_states._tmp`) is passed
  - The overall number of tracks in passed (`track_states._count`) as one call to OpenCL performs the Similarity operation for all tracks

```
compute_context.m_similarity.apply( predictions._prediction,
                                    track_states._err,
                                    track_states._tmp,
                                    track_states._err_new,
                                    track_states._count);
```

All Tracks at the same time