Matrix Element Method for High Performance Platforms

Principal author : **G.Grasseau***.

Co-author & speaker : **D.Chamont***.

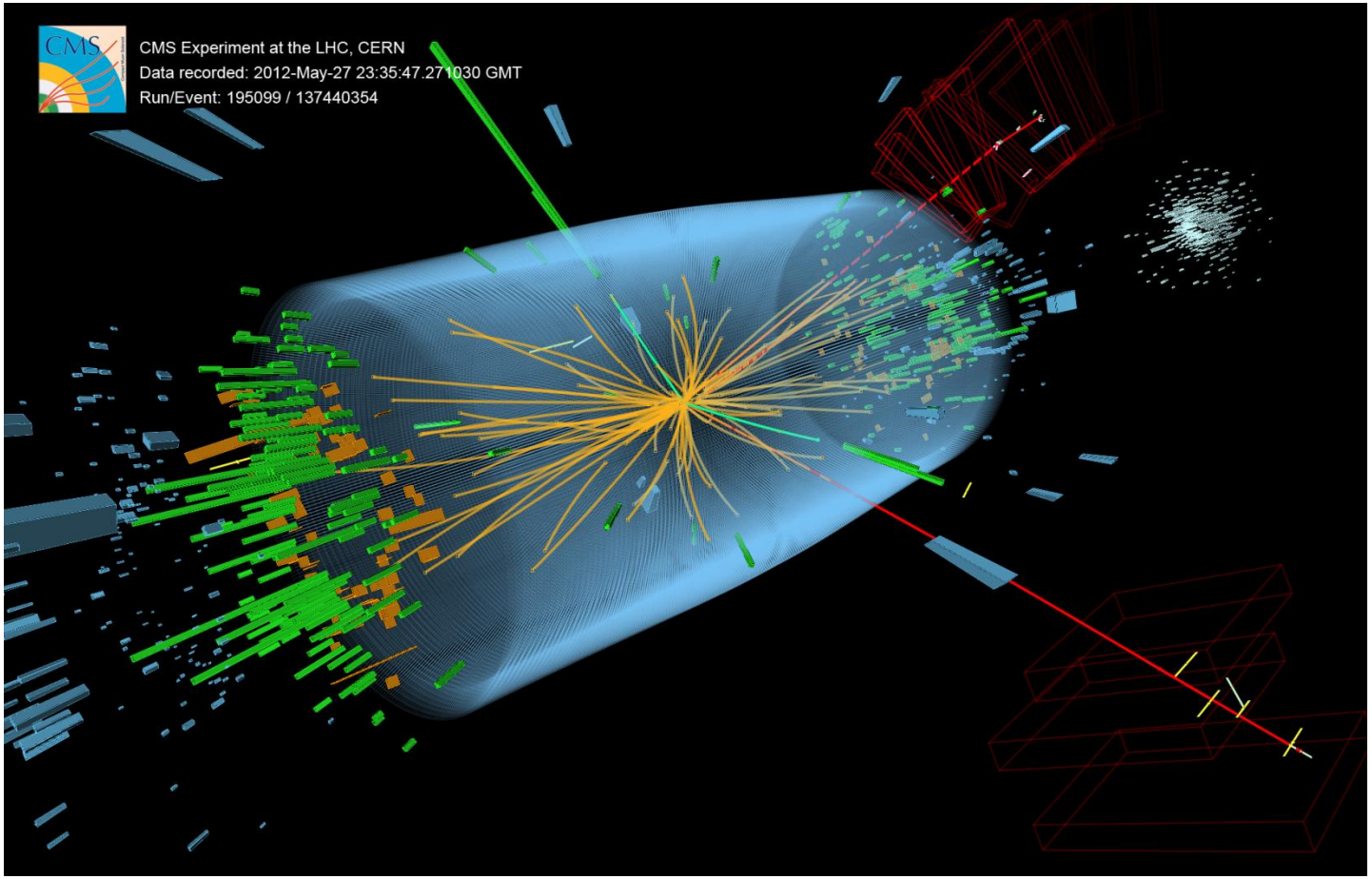
Co-authors: **F.Beaudette*, L.Bianchini⁰, O.Davignon*, L.Mastrolorenzo*, T.Strebler*.**

** Laboratoire Leprince-Ringuet*

⁰ ETH Zurich

CHEP, 16 April 2015

Use-case : CMS H→ττ analysis



Use-case : CMS H→ττ analysis

- The Matrix Element Method is an analysis method trying to exploit optimally all the information of the event.
- For each event (data or simulated), we want to compute a weight quantifying the probability that it arises from a given theory model (here H→ττ) :

observed event
theory model
first parton impulsion fraction
second parton impulsion fraction
real event

$$P(\mathbf{x}|\Omega) = \frac{1}{\sigma_\Omega} \iiint dx_1 dx_2 d\mathbf{y} \mathcal{P}_s(x_1, x_2) |\mathcal{M}_\Omega(x_1, x_2, \mathbf{y})|^2 W(\mathbf{x}, \mathbf{y})$$

parton density function
matrix element
transfer function (response of the detector)

- The ME method has never been applied to final states with tau leptons before.

Computing aspects

- Multi-dimensional integral (typically 5 for $H \rightarrow \tau\tau$)
  - To be computed with a MC approach (typically 20k points for 5 dimensions).
- Several different software packages involved :

$$P(\mathbf{x}|\Omega) = \frac{1}{\sigma_\Omega} \underbrace{\int\int\int} dx_1 dx_2 d\mathbf{y} \underbrace{\mathcal{P}_s(x_1, x_2)} \underbrace{|\mathcal{M}_\Omega(x_1, x_2, \mathbf{y})|^2} \underbrace{W(\mathbf{x}, \mathbf{y})}$$

ROOT GSL VEGAS | LHAPDF | MADGRAPH generated code | « hand-made »

- To be processed for **Run 1** (2010-2012, 7-8 TeV) :
  - 200k events, to be processed both for signal and background.
  - Estimated maximum time to process 1 event : 30 minutes... **~4100 days** for the whole run.
- Expected for the whole **Run 2** (2015-2018, 13-14 TeV) :
  - Same processing time per event.
  - **5x more** events.

➢ **Accumulate every computing resource available (nodes and devices), so to perform the analysis within an acceptable timeframe.**

Key technological choices

- Within a node, OpenCL: portable code which can benefit from all cpus and accelerators (from nvidia, amd, intel), and benefit from SIMD.
  - Big price to pay : porting most of the code to a subset of C99.
- Between nodes, MPI : can handle any kind of problem, embarrassingly parallel or not.
  - Currently, we mainly use MPI to distribute events.
  - For integrals with many dimensions, we study the distribution of subdomains of the integral.
  - Also helpful to dynamically control load balancing.

Two lines of development

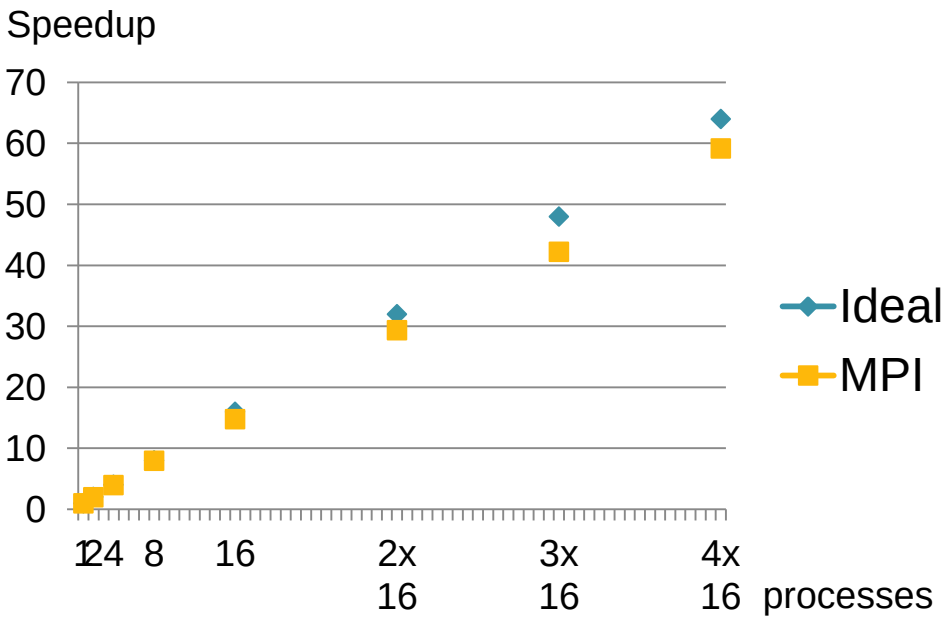
- The **MEM-MPI pre-production code** : contains all the physics for the channel $H \rightarrow \tau\tau$, all the glue to CMS data, but only MPI instrumentation. Driven by physicists.

- The **MEM-MPI-OCL prototype** : contains all the machinery to opportunely accumulate any processing hardware available, but only a subset of the physics has been ported to OpenCL. Driven by engineers.

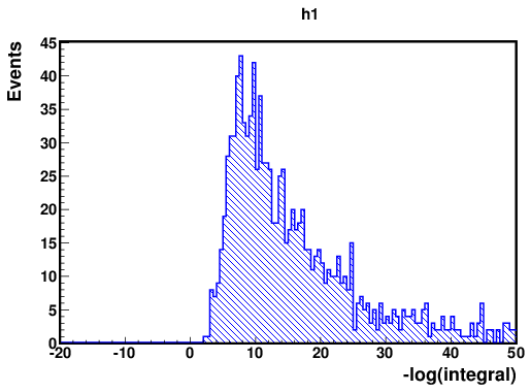
➢ Of course, the aim is to progressively improve both and merge them.

MEM-MPI : first results

- Computing time
  - Scales very well from 1 to 16 cores within a single node, and until 4x16 cores with 4 nodes.



- Physics results
  - Validated by physicists
  - Under certification with CMS official simulated and real data.



VBF simulated events under the VBF hypothesis

MEM-MPI-OCL : porting status

- The major task is the translation to OpenCL of all code involved (~30 kloc).
- Porting steps:
  1. From C++ & Fortran to C99.
  2. From C99 to OpenCL.
- Porting status:

$$P(\mathbf{x}|\Omega) = \frac{1}{\sigma_\Omega} \iiint dx_1 dx_2 d\mathbf{y} \mathcal{P}_s(x_1, x_2) |\mathcal{M}_\Omega(x_1, x_2, \mathbf{y})|^2 W(\mathbf{x}, \mathbf{y})$$

VEGAS	LHAPDF	MADGRAPH	Home-made
Done	***Almost Done***	***Ongoing***	***Started***
~2000 loc	~1600 loc	~3000 loc + ~ 6000 loc (signal) + ~10000 loc (background)	<1000 loc

<1000 loc (core software: event reading, kinematic transformations, etc....)

MEM-MPI-OCL : first results (CPU)

- Integration of a single Matrix Element (uu->h->uu tau tau).
- Reference for the speedup : GSL implementation running on 1 core.
- Benchmark hardware: 2 x E5-2650 => 16 cores, with AVX (4 doubles).
- Comments about the column “16 cores (MPI)”
  - We run 16 MPI processes.
- Comments about the column “16 cores (OCL)”
  - We run a single MPI process which see the motherboard as 16 OpenCL devices.
  - The important speedup comes from Intel OpenCL implementation which nicely takes advantage of CPU vectorization features (AVX).

	16 cores (MPI)	16 cores (OCL)
s/event	1,69	0,53
Speedup	**14,6**	**46,6**

MEM-MPI-OCL : first results (K20)

- Integration of a single Matrix Element (uu->h->uu tau tau).
- Reference for the speedup : GSL implementation running on 1 core.
- Benchmark hardware: 16 cores + 2 x K20M.
- Comments about the column “1xK20 (OCL)”
  - We run a single MPI process which delegates all computing to an NVidia K20 card.
  - Best speedup, more than 1.5 compared to previous 16 cores with OpenCL.
- Comments about the column “2xK20 (MPI-OCL)”
  - We need to start two MPI processes, because the NVidia OpenCL implementation does not support multiple devices.
  - Performance scales as expected.

	16 cores (OCL)	1xK20 (OCL)	2xK20 (MPI-OCL)
s/event	0,53	0,32	0,18
Speedup	**46,6**	**77,1**	**140,5**

MEM-MPI-OCL : first results (PHI)

- Integration of a single Matrix Element (uu->h->uu tau tau).
- Reference for the speedup : GSL implementation running on 1 core.
- Benchmark node : 16 cores + 2 x Phi 5110P.
- Comments about the column “1xPhi (OCL)”
  - We run a single MPI process which delegates all computing to an Intel Xeon Phi card.
  - Slightly better than 16 cores with MPI, but largely below 16 cores with OpenCL (disappointing, this probably deserves more investigation).
- Comments about the columns “2xPhi (OCL)” and “16 cores +2xPhi (OCL)”
  - Intel OpenCL implementation nicely handle two devices plus the motherboard.
  - Performance scales as expected.

	1xPhi (OCL)	2xPhi (OCL)	16 cores+2xPhi (OCL)
s/event	*1,31*	*0,63*	*0,33*
Speedup	***18,9***	***38,9***	***74,2***

Extrapolation

- Estimation for Run 1 sequential analysis: ~4100 days.
- Assumptions
  - The K20 speedup (~77), obtained for a single madgraph process, will be the same with all madgraph processes.
  - Good MPI+OpenCL global multi-node load balancing.

➢ Processing the whole Run 1 should require
  ➢ Either ~**28 nodes of 16 cores running 10 days** (4100/15/10),
  ➢ or ~**6 K20s running 10 days** (4100/77/10).

- Favorable factors not taken into account
  - Also use the CPU cores together with the K20s (under work).
  - Profiling and optimization of OpenCL code (future work).
  - Factorizing the ME, without compromising on the physics, another factor 5 could be gained.

Conclusion & future work

- The mix of MPI and OpenCL proves efficient. No technical showstopper encountered.
- **With 6 high-end computing accelerators, we aim to run a whole Run1 analysis within 10 days.**
- Thanks to these improvements tuned on Run1, applying MEM on Run2 data can be envisioned.
- Future work
  - Priority : finalize OpenCL porting and merge codes.
  - Then : profile the execution and improve implementation.
  - Work on the normalization integral
    - Independent of event data, but more dimensions (about 10).
    - We plan to use MPI so to distribute subdomains of the integral.
  - Investigate OpenCL-CUDA bridges

The MEM-HPC team

- CMS Physicists (LLR - Ecole polytechnique)
  - Florian Beaudette, Olivier Davignon, Luca Mastrolorenzo, Thomas Strebler, Pascal Paganini.
- Engineers (LLR - Ecole polytechnique)
  - Gilles Grasseau, David Chamont.
- External Collaborators
  - CMS MEM : Lorenzo Bianchini (ETH Zurich).
  - OpenCL-CUDA bridge : Gergely Debreczeni (Wigner RCP), http://gpu.wigner.mta.hu/.

This work has been funded by the P2IO LabEx (ANR-10-LABX-0038) in the framework « Investissements d'Avenir » (ANR-11-IDEX-0003-01) managed by the French National Research Agency (ANR).

Backup slides

About CPU hyperthreading

- We have confirmed that hyperthreading, supposed to benefit to non-optimized applications, generally does not improve our performances.
- With MEM-MPI preproduction code, running more MPI processes that the real number of CPU cores (16) can reduce the throughput :

Nb MPI processes	8	16	24	32
Events/s	0,082	0,152	0,148	0,166
Speedup	**7,97**	**14,78**	**14,42**	**16,17**

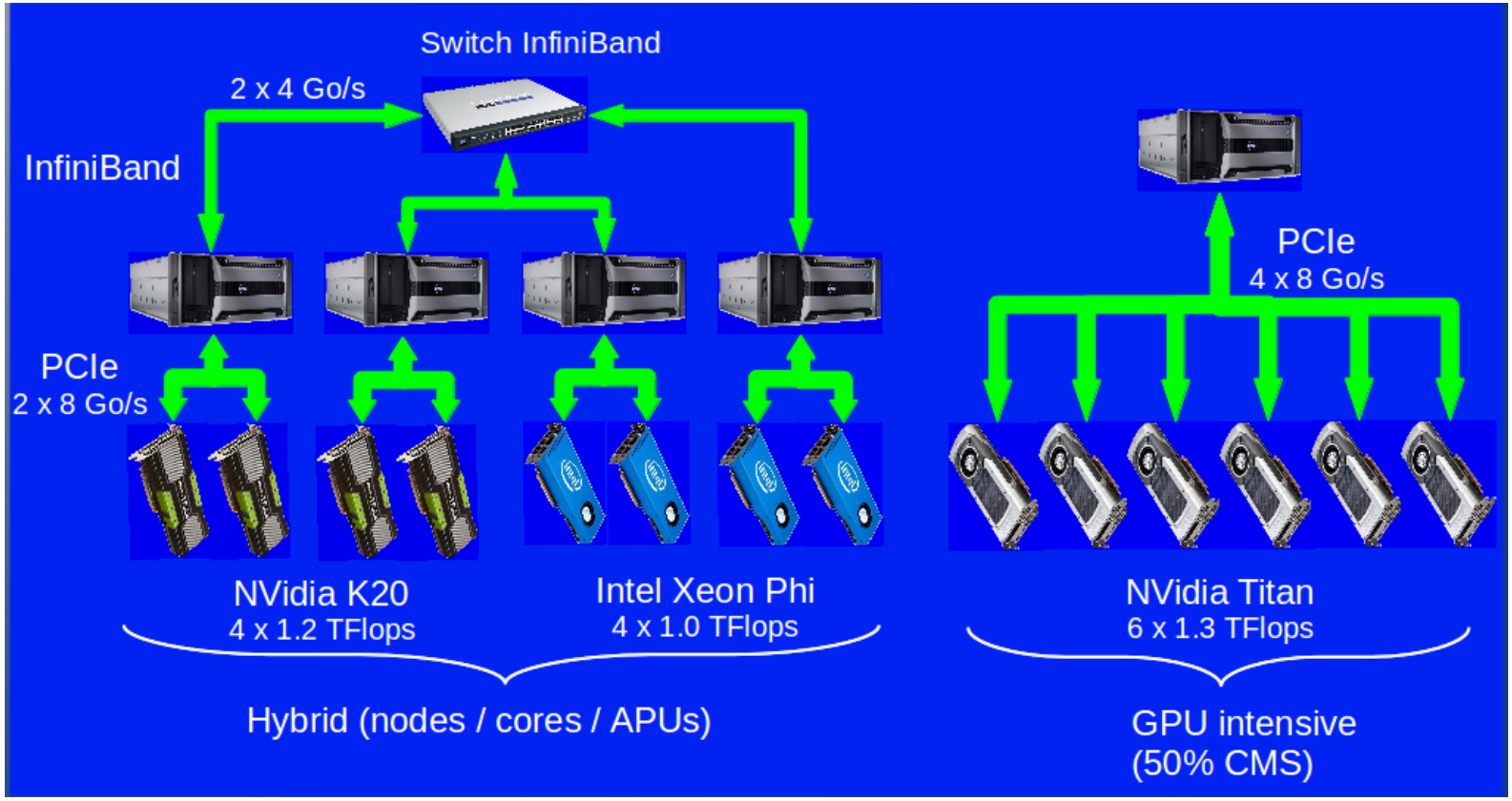
- With MEM-MPI-OCL prototype, not using OpenCL, we see an improvement :

Nb MPI processes	16	32
Events/s	0,59	0,69
Speedup	**15,6**	**18,19**

- With MEM-MPI-OCL prototype, using OpenCL for the CPUs, hyperthreading does not affect the throughput.

Benchmark platform

(GridCL, funded by P2IO)



Software : **OpenCL** AMD & Intel**, CAPS Compiler (**OpenHMPP et **OpenACC)**, OpenMPI, **Cuda-5.5**, **Intel Cluster Studio XE 2013** (C++ Compiler, Fortran Compiler, Integrated Performance Primitives, Threading Building Blocks, Math Kernel Library, MPI Library, Trace Analyzer and Collector, VTune Amplifier, Inspector, Advisor)