



Science & Technology  
Facilities Council

# Summary of CERN Workshop on Future Challenges in Tracking and Trigger Concepts

*Abdeslem DJAOUI*

**PPD**

**Rutherford Appleton Laboratory**



## Background to Workshop

- Organised by CERN OpenLab and held in IT Auditorium, 7-8 July 2011
- ~30 registered participants
  - Mainly from heavy-ion interaction experiments
- Follow up to a first Workshop in June 2010 held in GSI
- Aim:
  - How exploit new processors (CPUs, co-processors) in order to reduce cost and complexity as well as speeding up triggers and analysis software?
- Two tracks
  - First day: Algorithms
    - Reconstruction methods (track finding/fitting), ...
  - Second day: Parallel Hardware and software
    - Tools for parallelization/Vectorization, new Intel MIC architecture, ...



# First Workshop and Context

- Targeted experiments
  - GSI CBM (Compressed Baryonic Matter)
  - BNL STAR (Relativistic Heavy Ion Experiment)
  - PANDA (Darmstadt) and ALICE (CERN)
- Consolidated effort involving:
  - GSI: Algorithms
  - Uni Frankfurt: Vector Classes
  - HEPHY (Vienna): Kalman Filter, track and vertex fit
  - Intel: Intel Ct (C/C++ for throughput computing)
  - CERN OpenLab: Many core optimization, benchmarking
- Topics
  - Vertexing, Track fit, Track finding
  - Parallelization on CPUs and GPUs



## Next slides will summarize each talk

- Thursday: Algorithms
  - 1) Summary of progress since 1<sup>st</sup> workshop
  - 2) GPUs for triggering in the NA62 experiment
  - 3) Algorithms challenges in LHC data analysis
  - 4) Random number generation for MC simulations using GPUs
  - 5) New approaches in Track finding/fitting
  - 6) KFParticle and a Vertexing toolkit
  - 7) Experience using GPU for ATLAS Z-finder
- Friday morning: Hardware/Software
  - 8) ArBB: Software for scalable programming
  - 9) OpenCL: programming at the right or wrong level
  - 10) Intel MIC accelerator
  - 11) NUMA experience on large many-core servers
  - 12) How to improve (in the HEP community)



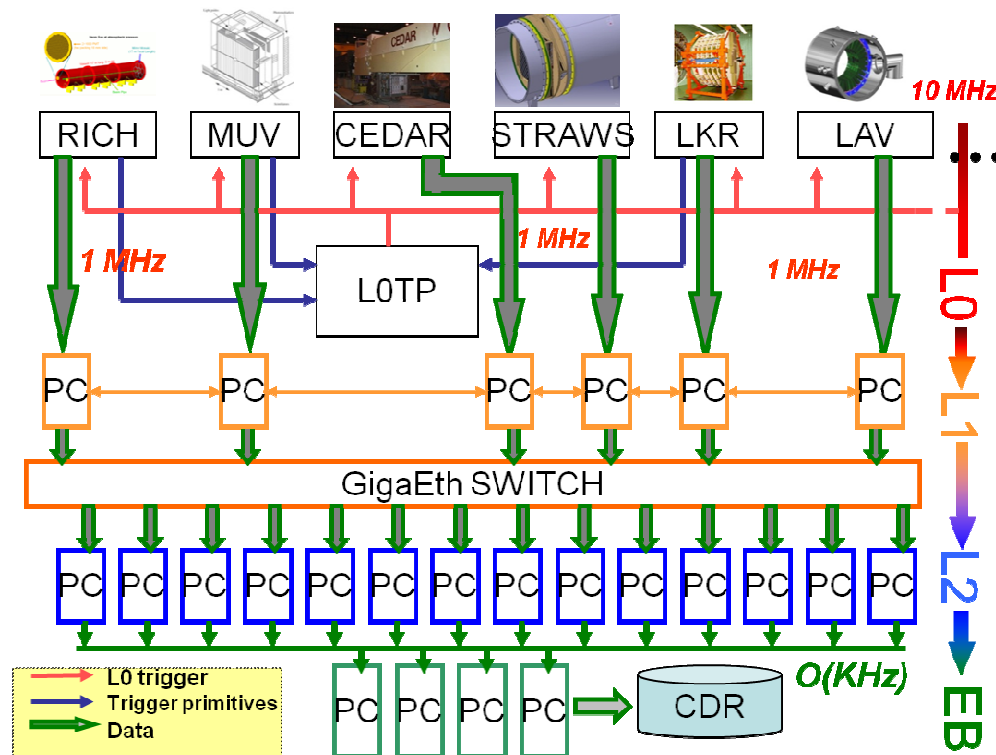
## 1) Progress Since First Workshop

- Vertexing
  - 3 D model under development
- KFParticle package extended with some routine
  - KFParticle functionality:
    - Reconstruction of decayed particle parameters based on Cellular Automaton (CA) track finder, Kalman Filter (KF) track fitter (More details in talks 5 and 6)
    - Derived from cbmroot framework
- Track fit
  - Improved stability and speed of track fitting
  - Single precision Runge-Kutta track propagation implemented
- Track finding
  - Efficiency and speed improves on CBM and STAR experiments
- Track propagation in magnetic field
  - 40 X speed up on GPUs demonstrated



## 2) TDAQ in NA62 Experiment

NA62 : to probe decay of charged Kaons, using 400 GeV/c protons

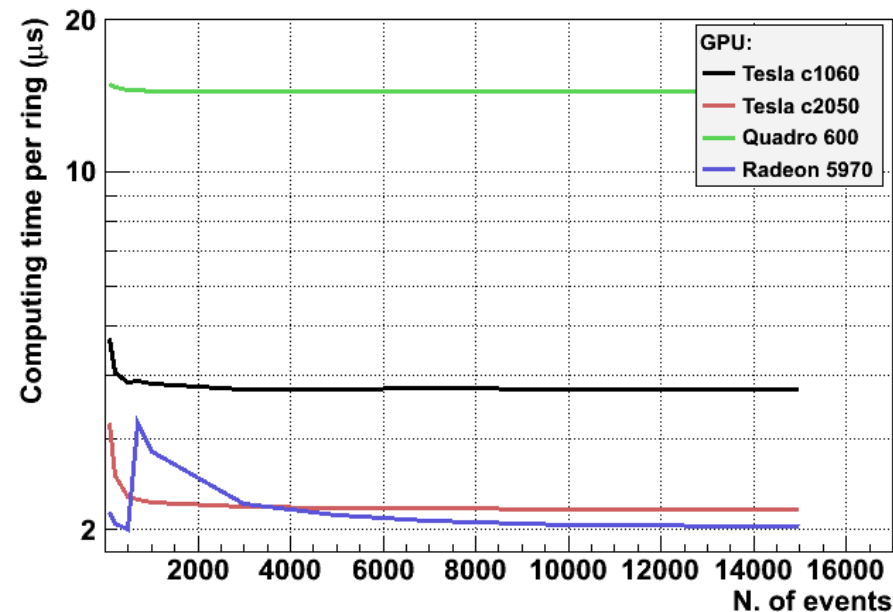


- L0: Input rate is ~10 MHz. latency ~1ms. (~2 microseconds for ATLAS)
- L1: Input rate ~1 MHz. latency is few seconds. (few ms for ATLAS)



## GPUs in NA62 Triggers (1/2)

- The use of the GPU at the software levels (L1/2) is straightforward: just put the video card in the PC!
  - No particular changes to the hardware are needed
  - The main advantages is to reduce the number of PCs in the L1/L2 farms
- Various parallel algorithms for ring searches were described
- Various GPU cards tested
  - See graph
- Ring finding in the RICH of NA62
  - ~1.5 us per event @L1 using a new algorithm (double ring)





## GPUs in NA62 Triggers (2/2)

- The use of GPU at L0 is more challenging:
  - Need very fast algorithm (high rate)
  - Need event buffering in order to hide the latency of the GPU and to optimize the data transfer on the Video card
    - Limited by size of L0 buffer
- Ring finding in the RICH of NA62
  - ~50 ns per ring @L0 using a parallel algebraic fit (single ring)
- As a further application, investigating the benefit of GPUs for Online track fitting at L1, using cellular automaton for parallelization





## 3) Algorithm Challenges in the LHC Data Analysis

- High level general discussion on HEP data analysis
- Parallelization of RooFit using OpenMP required no drastic changes to existing code
  - RooFit: Library for modeling expected distribution of events
  - OpenMP: API for shared memory multi-process/multi-core thread programming
  - This Work is targeting multicore laptops and Desktop
    - ~2.5 X speed up without vectorization
    - Using Intel Compiler for vectorization (~5 X speed up)
  - Currently using double precision, but would like to use single precision
    - Can gain up to 2 speed up from vectorization
    - But numerical accuracy might be an issue



## 4) Pseudo Random Number Generation for MC Simulation Using GPUs

- Mersenne Twister method used in standard ROOT
  - Very good statistical properties with a period  $2^{19937}$
  - not suitable for GPUs
    - Has a large state that must be updated serially
- Hybrid Approach more suitable: Tausworthe and LCG (Linear Congruential Algorithm)
- Test example: multivariate alias sampling
  - Device: NVIDIA GeForce GTX 480
  - Kernel 1000 x times faster than CPU
  - Overall execution time only 10X (device-host data transfers)

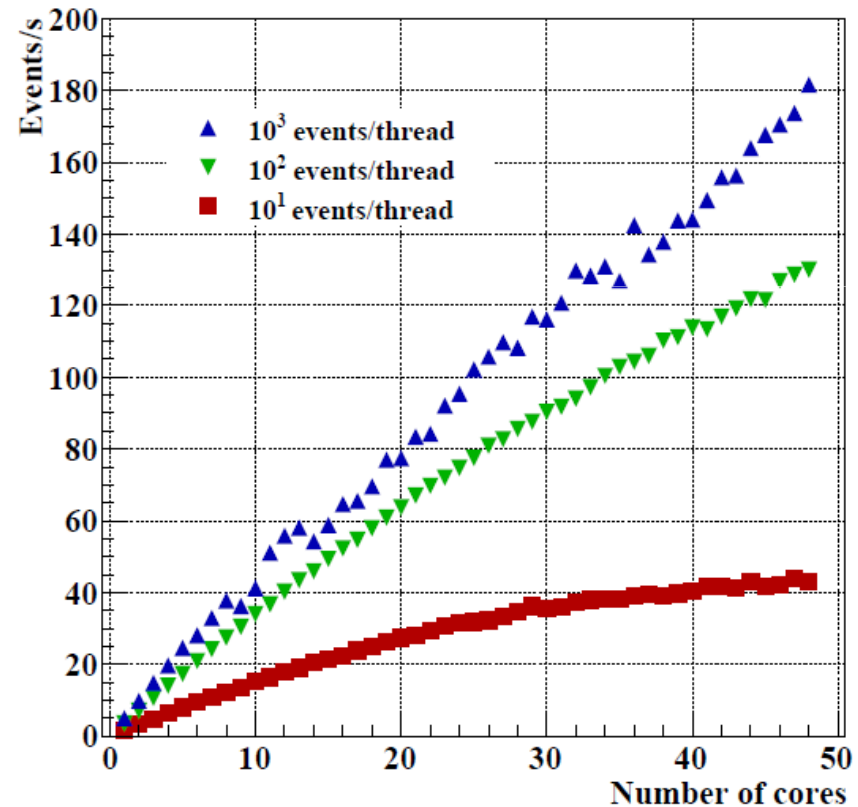
TABLE IV: The comparison of execution time

	CPU	GPU-CUDA	GPU-OpenCL
Kernel Exec.	1.202 Sec	53 $\mu$ sec	220 $\mu$ sec
Total time	1.402 sec	0.14 sec	0.146 sec



## 5) New approaches in Track finding/fitting

- Challenges in CBM
  - $10^7$  Au-Au col/sec
  - 1000 particles/col
- At L1
  - Track reconstruction
  - Vertex search
- Cellular Automaton CA as track finder
  - Ideal for many core CPUs and GPUs
  - Using Opladev35 node:
    - 4 CPUs AMD E6164HE, 12 cores/cpu



Larger groups of events per thread use the CPU more efficiently



## Kalman Filter Based Track Fit

- Discussed a number of approaches and track propagation method
- Similar many core scalability to CA
  - For 1000 tracks/thread, achieved 22 ns/track/node on opladev35
- The use of Intel ArBB for reconstruction algorithms in the STAR experiment is under investigation
  - ArBB: Array Building Blocks Library (Talk 8)
- 4D reconstruction (x, y, z,t ) for the CBM experiment has started



## 6) KFParticle and a Vertexing Toolkit

- KFParticle developed based on ALICE and CBM experiments
- Rewritten using SIMD model (Vectorization)
  - Vector arguments to some functions + for loops
- 5X speedup for CBM on multicore CPUs
- Online particle finder for CBM
  - Based on SIMD KFParticle and SIMD KF track fitter
  - Comparison with off-line model: practically the same results
- Future plans also discussed
  - Use KFParticle for 4D tracking development



## 7) Experience Using GPUs for ATLAS Z Finder Algorithm

- Introduction to GPUs, CUDA and GPU projects at Edinburgh
- Implementation of Z finder algorithm on GPUs
- Kalman Filter for CUDA in ATLAS
  - Edinburgh feasibility studies
  - 3 Slides on Dmitry's work
- Test case studies of particle tracking in a magnetic field, on GPUs
  - Preliminary results show speedup of 10 compared to CPU
- Particle tracking and simulation, in addition to trigger are areas where GPUs can used to improve performance




## 8) ArBB: Software for Scalable Programming

- ArBB: Array Building Blocks
  - C++ library for data parallel programming on many core CPUs, SIMD units and Intel MIC
    - Containers and parallel operations
      - Vectors and Matrices (regular , irregular, sparse)
  - ArBB programs cannot create thread deadlocks or data races
- Intel Compiler vectorization switches discussed
- ArBB is part of Intel Parallel Building Block:
  - Intel Cilk Plus
  - Intel Threading Building Block
  - Intel ArBB



# Intel Parallel Building Blocks

Intel® Parallel Building Blocks 			
	Intel® Cilk™ Plus	Intel® Threading Building Blocks	Intel® Array Building Blocks
What is it?	Language extensions to simplify task and vector parallelism	Widely used C++ template library for task parallelism	Sophisticated C++ template library for vector parallelism
Features	<ul style="list-style-type: none"><li>▪ 3 Simple keywords and array notations for parallelism</li><li>▪ Support for Task and Vector parallelism</li><li>▪ Similar semantics as serial code</li></ul>	<ul style="list-style-type: none"><li>▪ Parallel Algorithms and Data Structures</li><li>▪ Scalable Memory Allocation and Task Scheduling</li><li>▪ Synchronization Primitives</li></ul>	<ul style="list-style-type: none"><li>▪ Automatically scales to future Intel platforms</li><li>▪ Use of cores, threads, SIMD determined by run time compiler</li></ul>
Reasons to Use	<ul style="list-style-type: none"><li>▪ Simple way to parallelize your code</li><li>▪ Sequentially consistent + low overhead = powerful solution</li><li>▪ Supports C &amp; C++; Windows* and Linux*</li></ul>	<ul style="list-style-type: none"><li>▪ Rich feature set for general purpose parallelism</li><li>▪ Available as open source or commercial</li><li>▪ Supports C++; Windows, Linux, Mac OS*, other OS's</li></ul>	<ul style="list-style-type: none"><li>▪ Used for flexible vector parallelism</li><li>▪ JIT &amp; VM technology = flexible and powerful</li><li>▪ Supports C++; Windows &amp; Linux</li></ul>

**MIX AND MATCH TO OPTIMIZE YOUR APPLICATION'S PERFORMANCE**





## 9) OpenCL: Programming at the Right or the Wrong Level

- Experience with a RooFit implementation
  - OpenCL is superset of C (no C++), cannot call host code from OpenCL
  - Neither Intel or AMD OpenCL implementations offer auto-vectorization on CPUs (as of 1/7/11)
    - Planned
  - Using C++ for CPU and OpenCL for GPU
  - OpenCL leads to larger serial fraction as compared to OpenMP
    - Needs explicit call to Kernel , lower parallel efficiency
- OpenCL can be painful for use in legacy C++ code
  - Main advantage is code reuse between CPU and GPU
  - Need to use double precision. Peak performance? Dream on!
- Think before using OpenCL for CPU
  - Intel compiler (or GCC) could be a better way to go
- Now investigating hybrid solution with OpenMP and OpenCL



## 10) Intel MIC Architecture

- MIC: Many Integrated Cores
  - X86 compatible multiprocessors that are programmed using existing (CPU) tools
  - Eliminates the need for dual programming models
  - Has all advantage of other accelerators for parallel workload
  - Each core is smaller, has lower power limit and executes its own instructions allowing complex code including branches, recursions, ...
- Prototype (Night's Ferry) available through 2011:
  - Up to 32 cores, 1.2GHz each
  - Up to 128 threads (4 threads/core)
  - Up 8 MB shared coherent cache
  - Up to 2GB high bandwidth GDDR5 memory
  - Only support single precision for now
- Commercial (Night's Corner) late 2012-2013
  - More than 50 cores per chip expected



## 11) NUMA Experience on Large Many-core Servers from AMD/Intel

- NUMA: Non Uniform Memory Access
  - Multiple nodes can access each other's memory but accessing local memory is faster
- Memory latency measurements
- Memory bandwidth measurements
- OpenMP on NUMA systems
  - Using STREAM bandwidth benchmark code
  - Compared the use of affinity switches (ICC compilers)
    - Compact: assign thread  $n+1$  as near as possible to thread  $n$
    - Scatter: distribute the threads as evenly as possible through system
      - Scatter can help performance
    - Useful to extract topology of the system for selection switch



## 12) How to Improve HEP Computing – expectations from people

- Control Numerical Accuracy through compiler, code, ...
  - $10^{-6}$  accuracy is fine,  $10^{-18}$  may be impossible
  - In the exa-scale era, you may not get the same result from two different runs using the same input on the same hardware, depending on the order in which floating point operations are interleaved
- Exploitation of vector capabilities
  - Every computer is now a vector computer
  - SIMD sizes (up to 16X for the MIC architecture, 512bits width)
    - Too much performance to ignore
- Which compiler to use
  - Look at parallelization/vectorization and accuracy control flags
- Use of parallelism with good control of memory
  - Rather incomprehensible that we use 1-10GB of memory to compute an event that is 1 MB in size
- Need to move all the good prototypes back into mainstream



## Final Thoughts

- Short afternoon discussion:
  - Many topics tabled, but limited debate
  - Action on making KFParticle package available to users
- Renewed activity in HEP analysis parallelization work
  - Vectorization seems to have a new lease of life at this workshop
    - Speed up can be achieved with minimal code changes
    - Standard programming models and compilers
- Intel MIC architecture will be another boost for vectorization and shared memory programming (OpenMP)
- GPUs have the edge in multi-level triggers (CA, KF)
  - But which GPU and which programming model for performance as well as avoiding vendor lock-in ?
- For more details see full slides:  
<http://indico.cern.ch/conferenceDisplay.py?ovw=True&confId=130322>