ICE-DIP:

> 23/04/2015

Przemysław Karpiński
CERN Openlab „ICE-DIP”

This research project has been supported by a Marie Curie Early European Industrial Doctorates Fellowship of he European Community’s Seventh Framework Programme under contract number (PITN-GA-2012-316596-ICE-DIP)”

Agenda

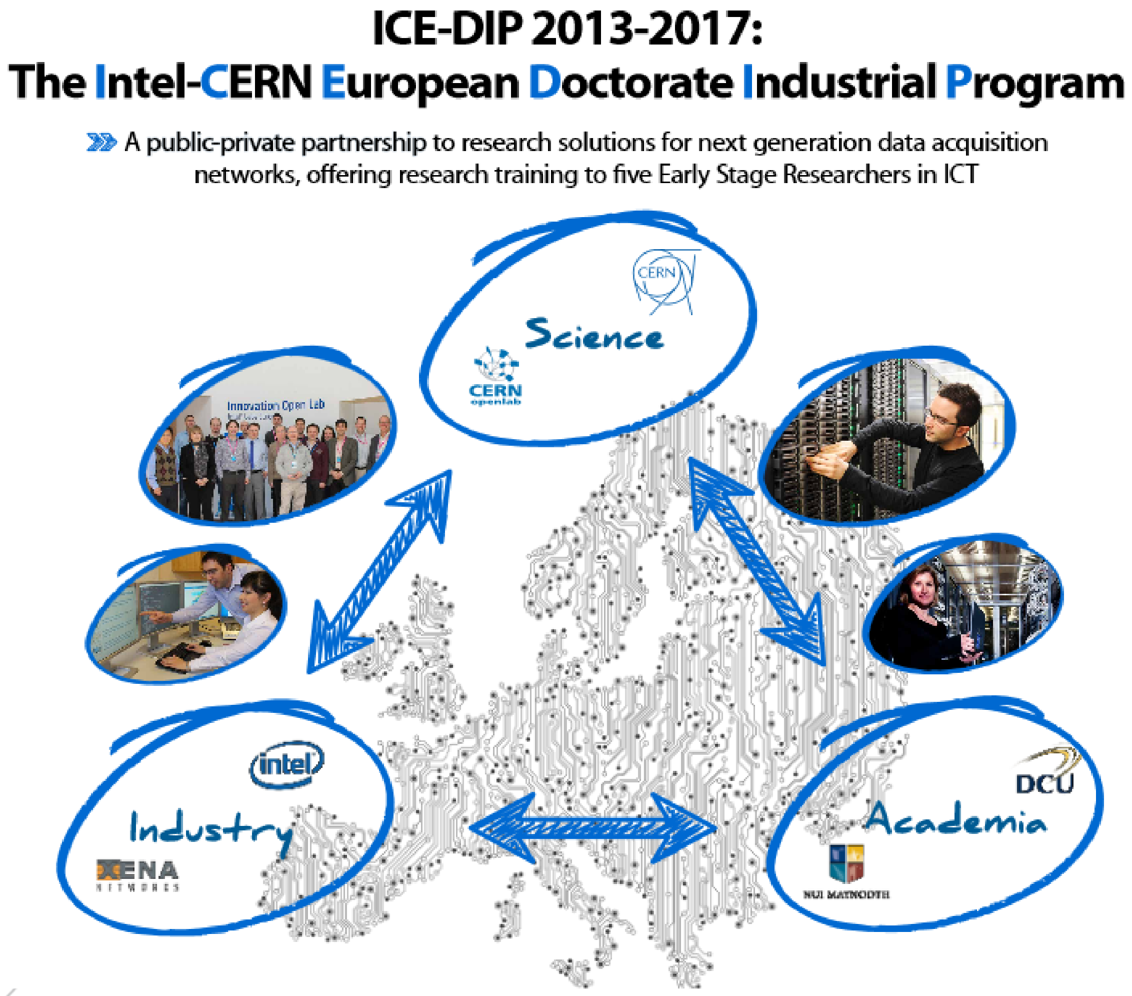
- ICE-DIP
- CERN Overview
- LHCb Triggered Data AcQuisition (TDAQ)
- Computing at CERN
- Explicit vectorization
- UME framework

ICE-DIP

ICE-DIP

Research topics:

- Silicon photonics systems
- Next generation data
- High speed configurable logic
- Computing solutions for high performance data filtering



ICE-DIP 2013-2017:
The Intel-CERN European Doctorate Industrial Program

A public-private partnership to research solutions for next generation data acquisition networks, offering research training to five Early Stage Researchers in ICT

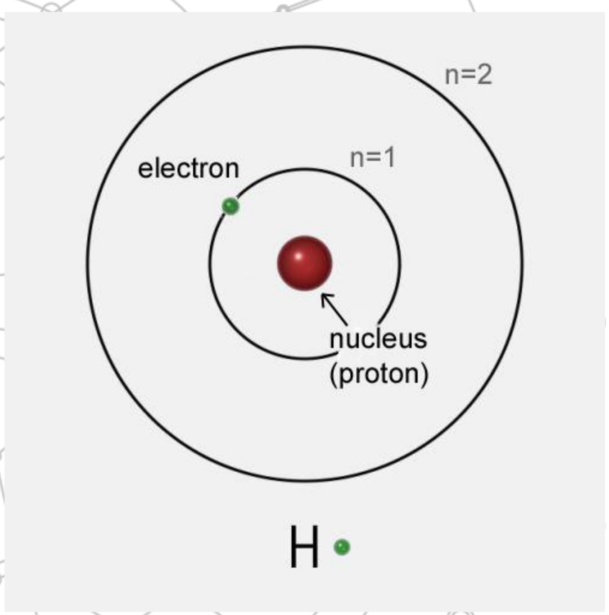
ICE-DIP Projects

Theme	WP	ESR	Challenge	Research
Silicon Photonics	WP1	ESR1 (Santa-clara, US)	Need affordable, high throughput, radiation tolerant links	Design, manufacture, test under stress a Si-photonics link
Reconfi-gurable Logic	WP2	ESR2 (Munich, Germany)	Reconfigurable logic is used where potentially more programmable CPUs could be proposed	A hybrid CPU/FPGA data pre-processing system
DAQ networks	WP3	ESR3 (Gdańsk, Poland	Bursts in traffic are not handled well by off-the-shelf networking equipment	Loss-less throughput up to multiple Tbit/s with new protocols
High performance data filtering	**WP4**	ESR4 (Munich, Germany)	Accelerators need network data, but have very limited networking capabilities	Direct data access for accelerators (network-bus-devices-memory)
		ESR5 (Paris, France)	**Benefits of new computing architectures are rarely fully exploited by software**	**Find and exploit parallelization opportunities and ensure forward scaling in DAQ networks**

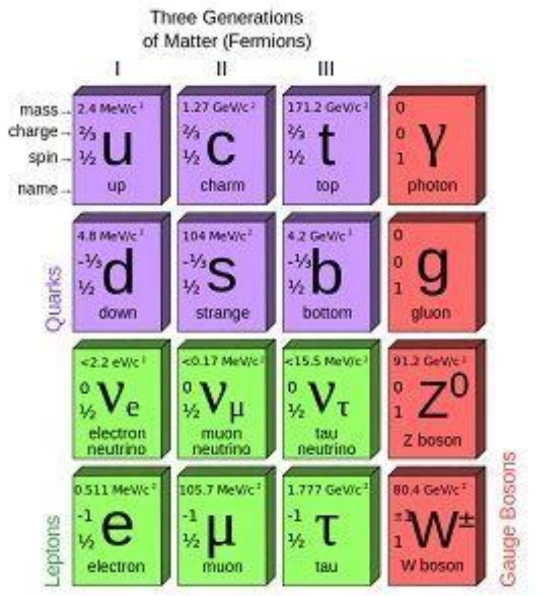
CERN Overview

Standard Model

BOHR MODEL



STANDARD MODEL



Three Generations of Matter (Fermions)

		I	II	III	
Quarks	mass / charge / spin / name	2.4 MeV/c² · ⅔ · ½ · u · up	1.27 GeV/c² · ⅔ · ½ · c · charm	171.2 GeV/c² · ⅔ · ½ · t · top	0 · 0 · 1 · γ · photon
Quarks		4.8 MeV/c² · -⅓ · ½ · d · down	104 MeV/c² · -⅓ · ½ · s · strange	4.2 GeV/c² · -⅓ · ½ · b · bottom	0 · 0 · 1 · g · gluon
Leptons		<2.2 eV/c² · 0 · ½ · ν_e · electron neutrino	<0.17 MeV/c² · 0 · ½ · ν_μ · muon neutrino	<15.5 MeV/c² · 0 · ½ · ν_τ · tau neutrino	91.2 GeV/c² · 0 · 1 · Z^0 · Z boson
Leptons		0.511 MeV/c² · -1 · ½ · e · electron	105.7 MeV/c² · -1 · ½ · μ · muon	1.777 GeV/c² · -1 · ½ · τ · tau	80.4 GeV/c² · ±1 · 1 · $W^\pm$ · W boson
					Gauge Bosons

CERN

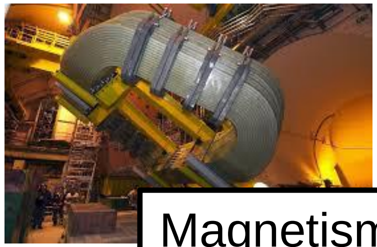
- HQ in Geneva (Switzerland)
- 61 years of existence
- 21 member states (Israel since 2014), 45 associate states, 17 cooperating states, 7 Observers
- ~14000 people

Background diversity

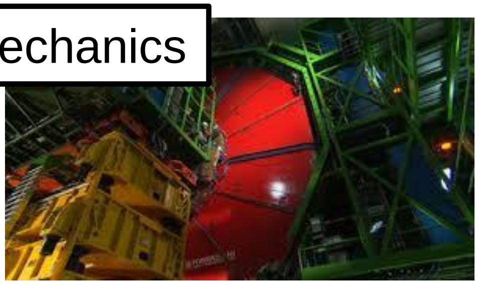
Computing/IT

Vacum, Criogenics

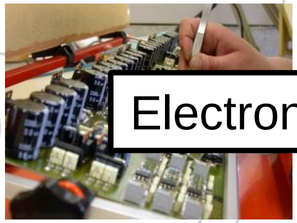
Magnetism



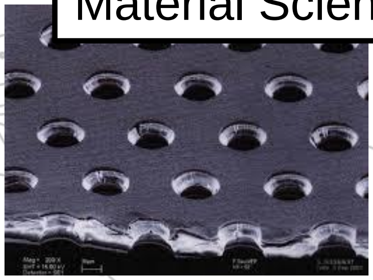
Mechanics



Electronics



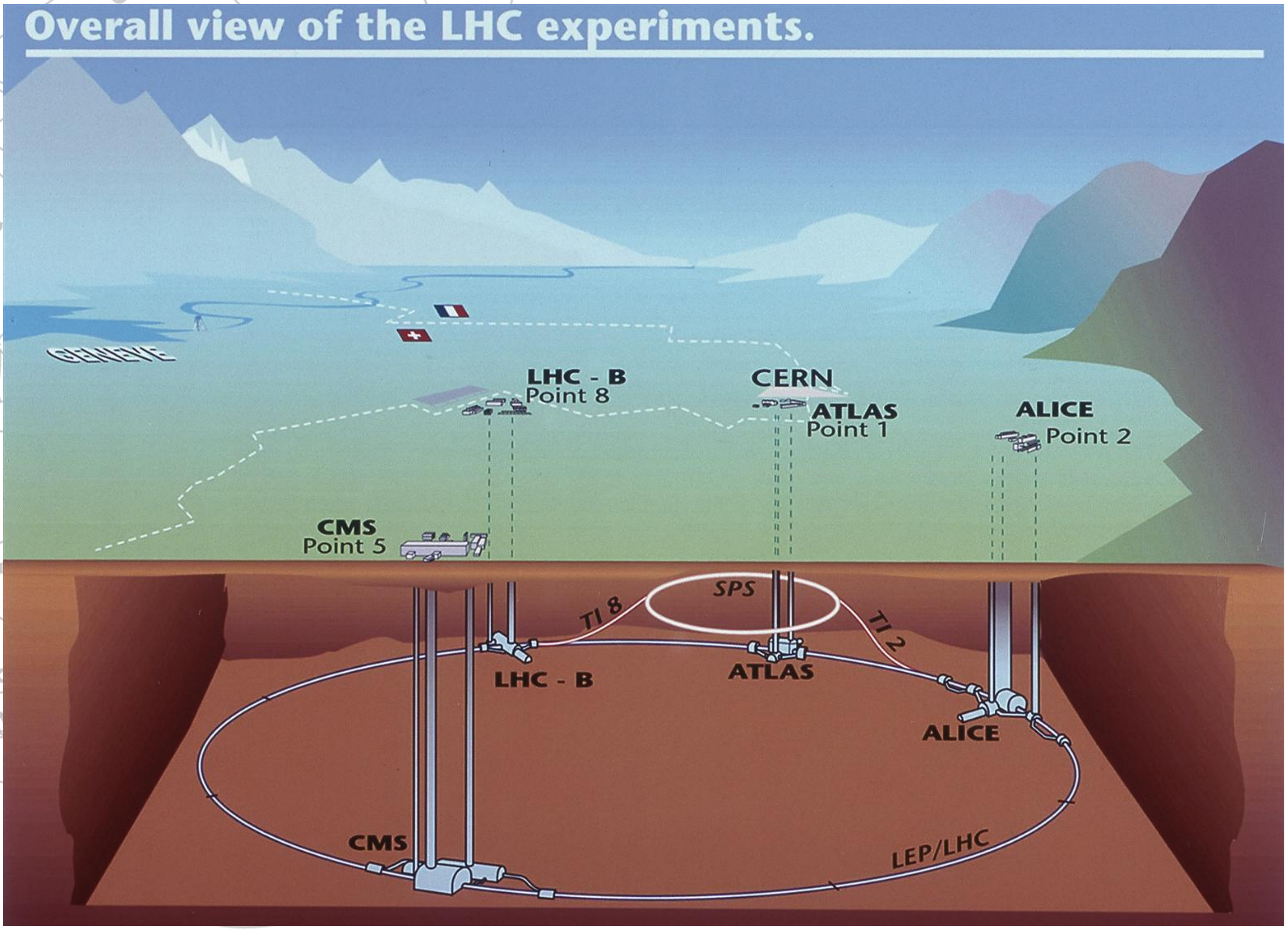
Material Sciences



Radiofrequency

Control systems

LHC


Overall view of the LHC experiments.
GENEVE
LHC - B
Point 8
CERN
ATLAS
Point 1
ALICE
Point 2
CMS
Point 5
SPS
TI 8
TI 2
LHC - B
ATLAS
ALICE
CMS
LEP/LHC

LHC

ATLAS: *A Thoroidal LHC Apparatus*

CMS: *Compact Muon Solenoid*

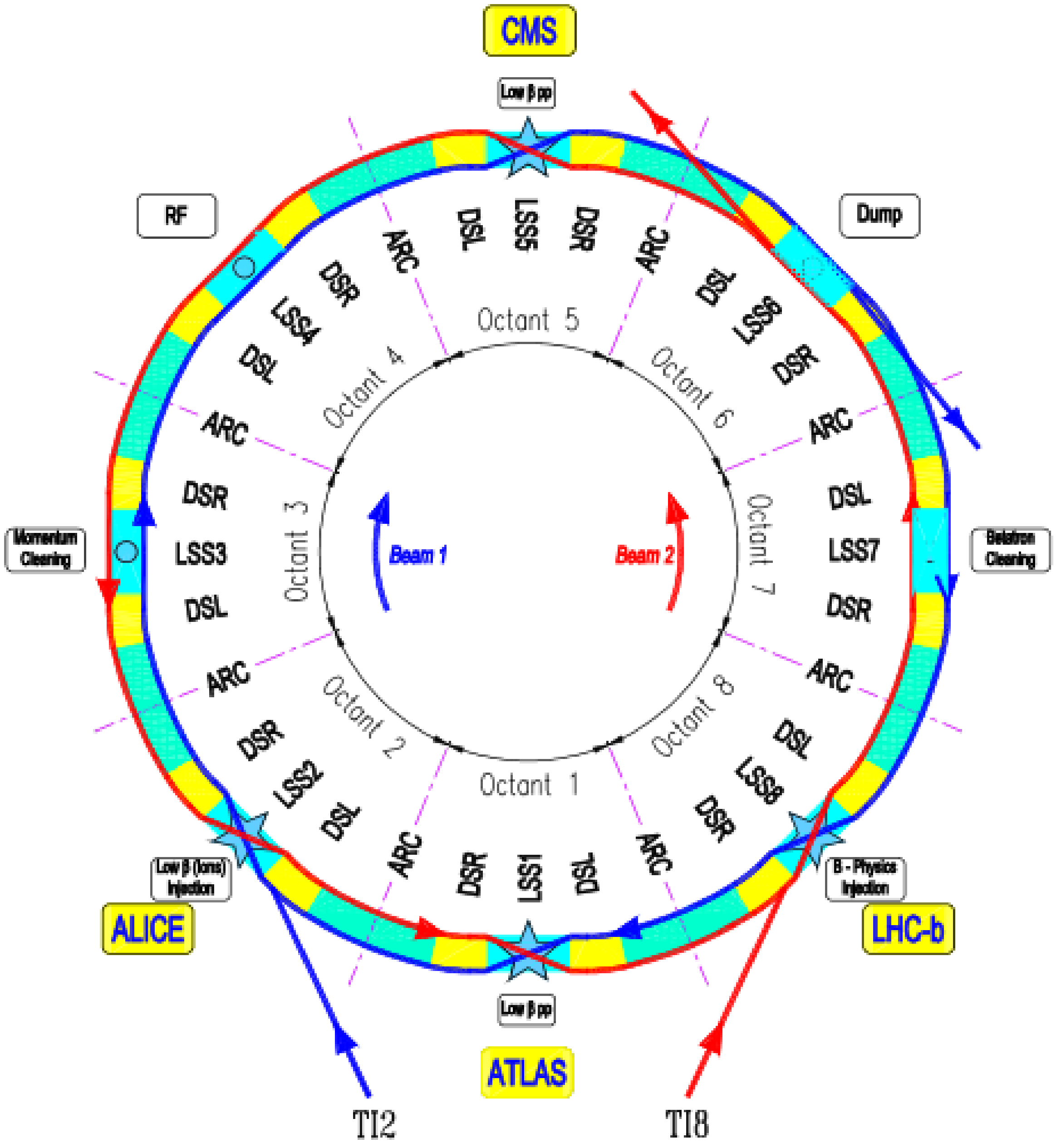
ALICE: *A Large Ion Collider Experiment*

LHCb: *Large Hadron Collider Beauty*

Eksperymenty:

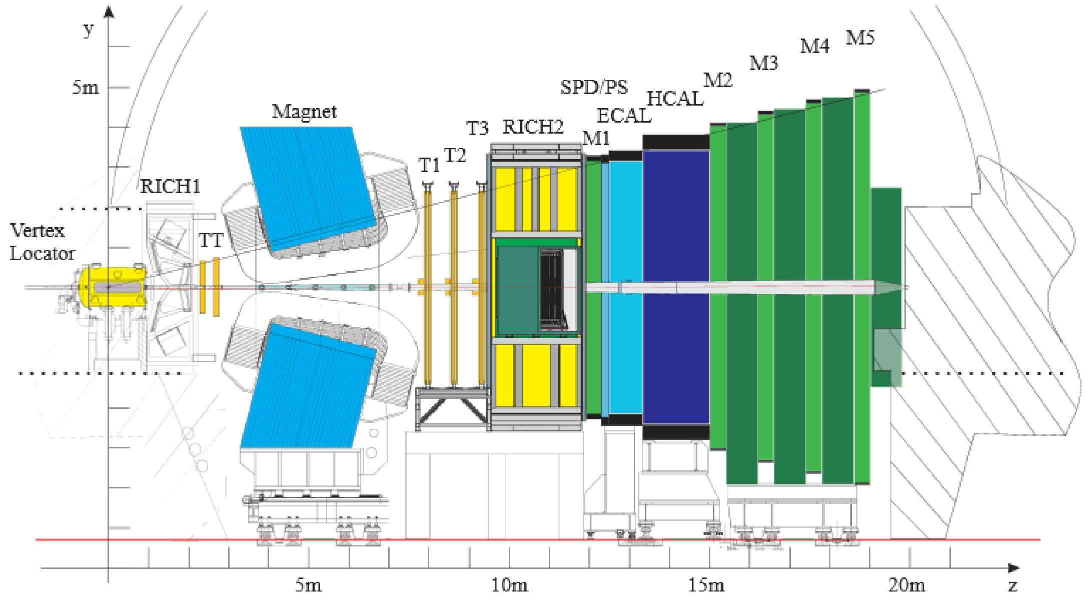
ACE, AEGIS, **ALICE**, ALPHA, AMS, ASACUSA, **ATLAS**, ATRAP, AWAKE, BASE, CAST, CLOUD, **CMS**, COMPASS, DIRAC, ISOLDE, **LHCb**, **LHCf**, **MOEDEL**, NA61/SHINE, NA62, NA63, nTOF, OSQAR, **TOTEM**, UA9

LHC


CMS
Low β pp
RF
Dump
Momentum Cleaning
Betatron Cleaning
Low β (Ions) Injection
B - Physics Injection
ALICE
LHC-b
Low β pp
ATLAS
TI2
TI8
Octant 1
Octant 2
Octant 3
Octant 4
Octant 5
Octant 6
Octant 7
Octant 8
Beam 1
Beam 2
LSS1
LSS2
LSS3
LSS4
LSS5
LSS6
LSS7
LSS8
ARC
DSL
DSR

LHC Trigger Data AcQuisition (TDAQ)

LHCb



VELO*:*
Collision point localization

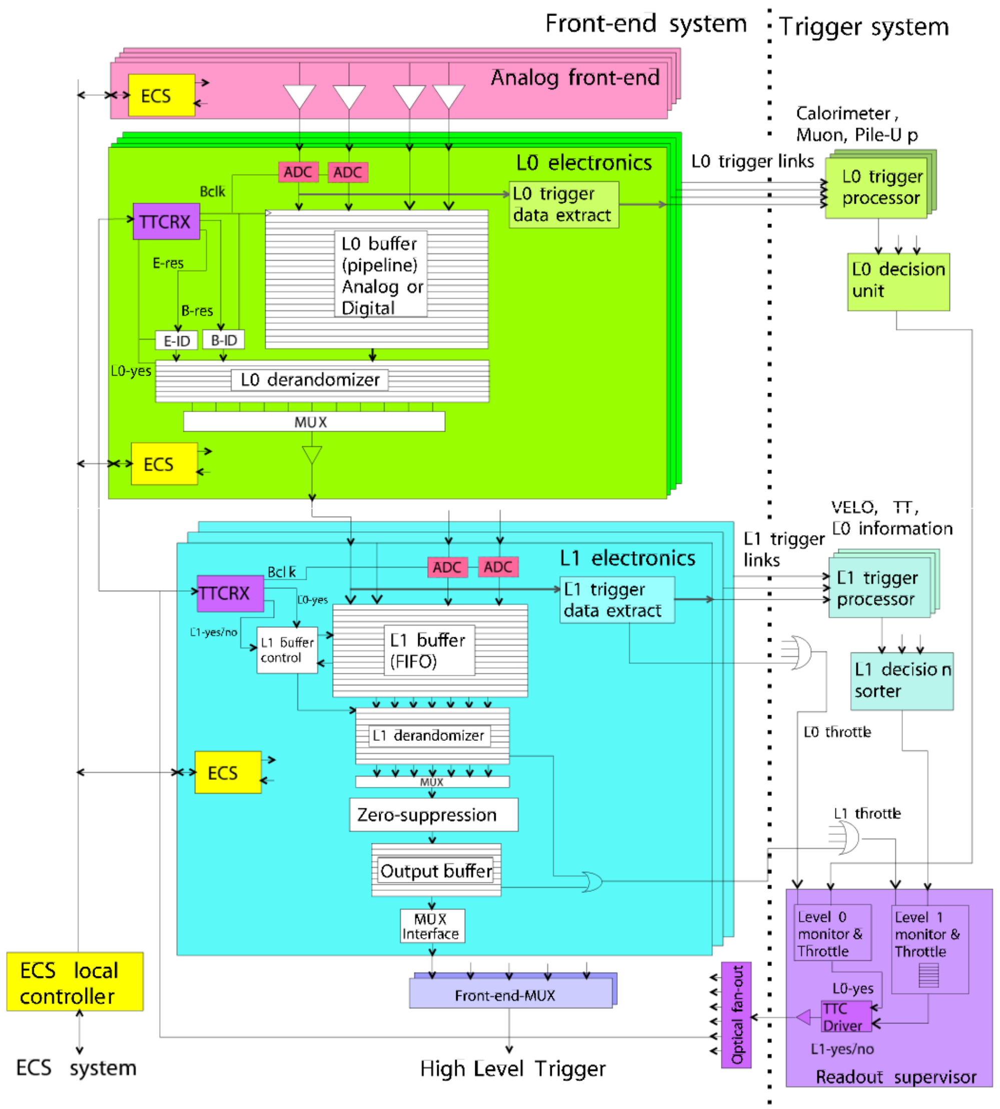
Inner/Outer Tracker:
Trajectories and momentum

RICH:
Particle identification

SPD, PS, ECAL, HCAL:
Hadron, electron, photon identification

MUON:
Particle identification

Trigger System



Tasks:

- Bandwidth reduction
- Data buffering

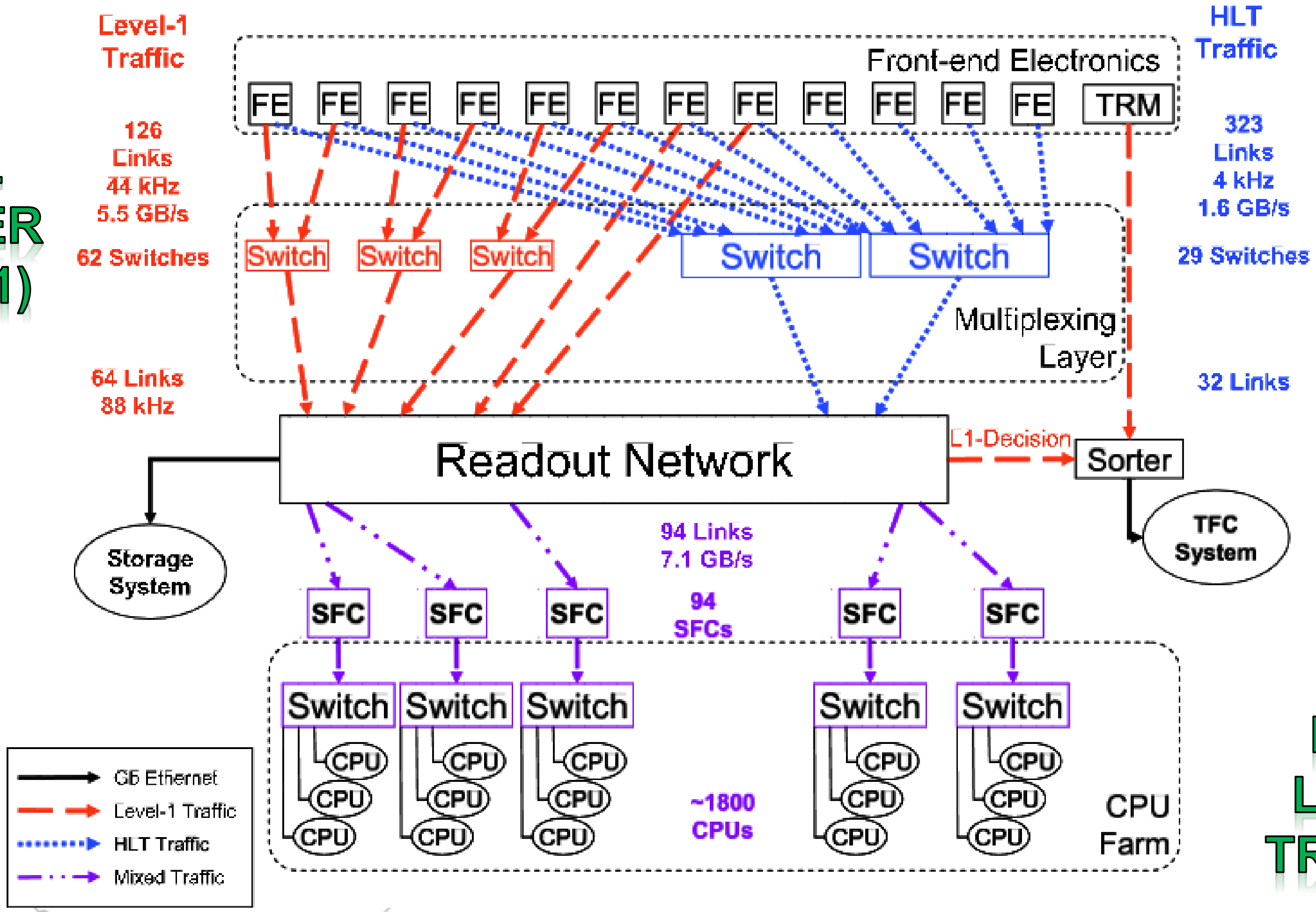
Some features:

- Hierarchic structure
- ASIC (L0)
- FPGA (L1)
- Non-standard solutions!

Level	Event Frequency	Bandwidth
Front-end	40MHz	4TB/s
L0	1MHz	100GB/s
L1	40kHz	4GB/s
HLT	400Hz	40MB/s

Readout Network

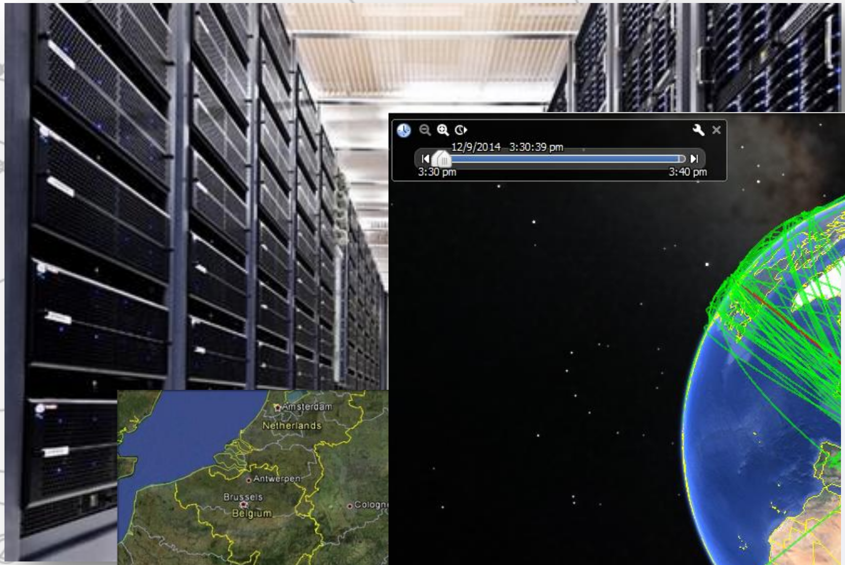
LOW LEVEL TRIGGER (L0 + L1)

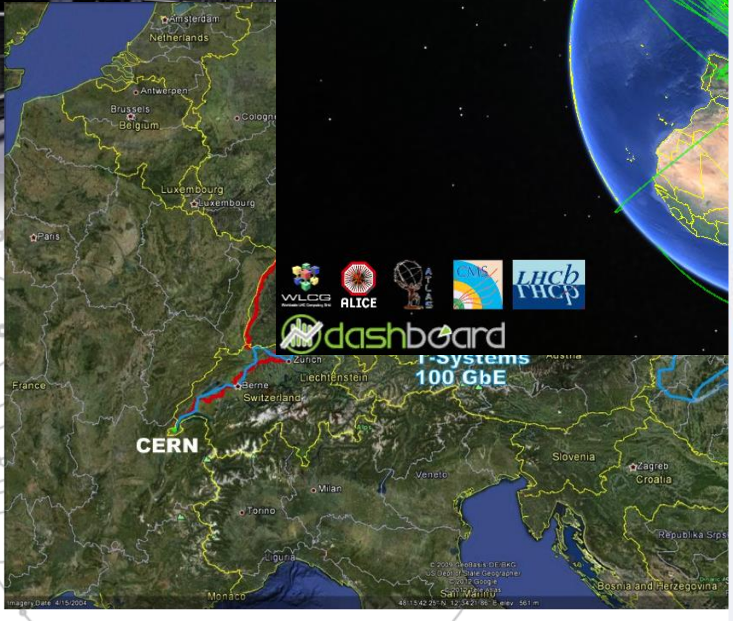


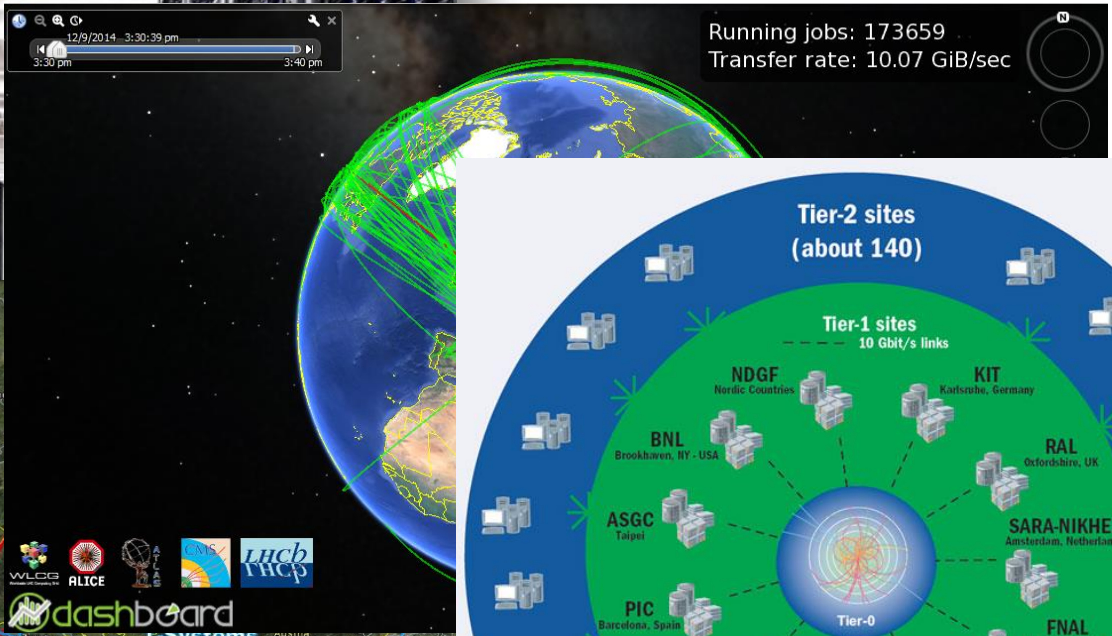
HIGH LEVEL TRIGGER (HLT)

Computing at CERN

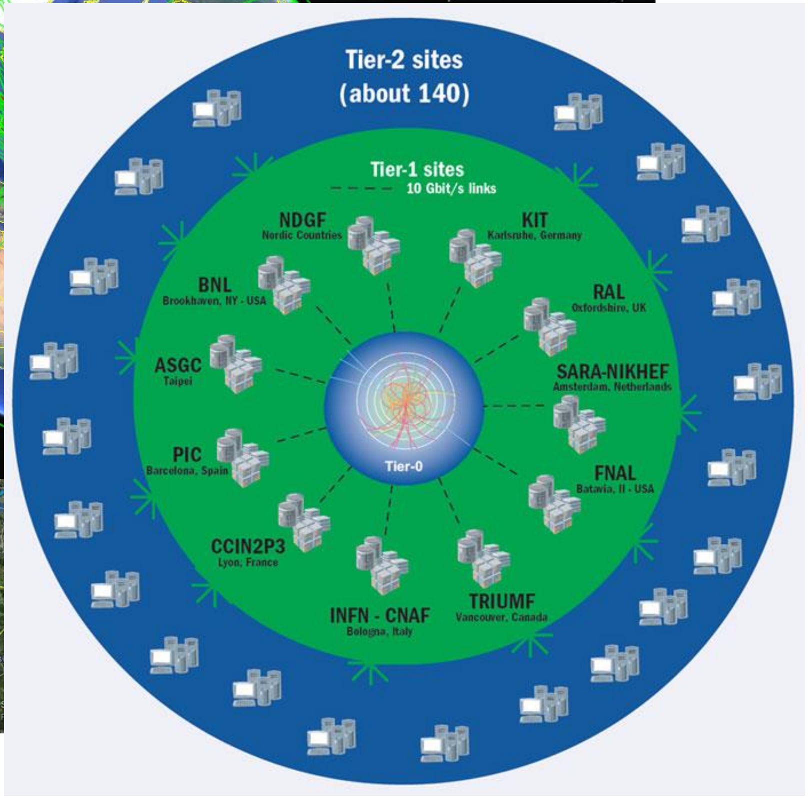
Computing at CERN







Running jobs: 173659
Transfer rate: 10.07 GiB/sec



CERN openlab

Computing at CERN

ONLINE:

Gaudi

WLCG
Worldwide LHC Computing Grid

ALFA

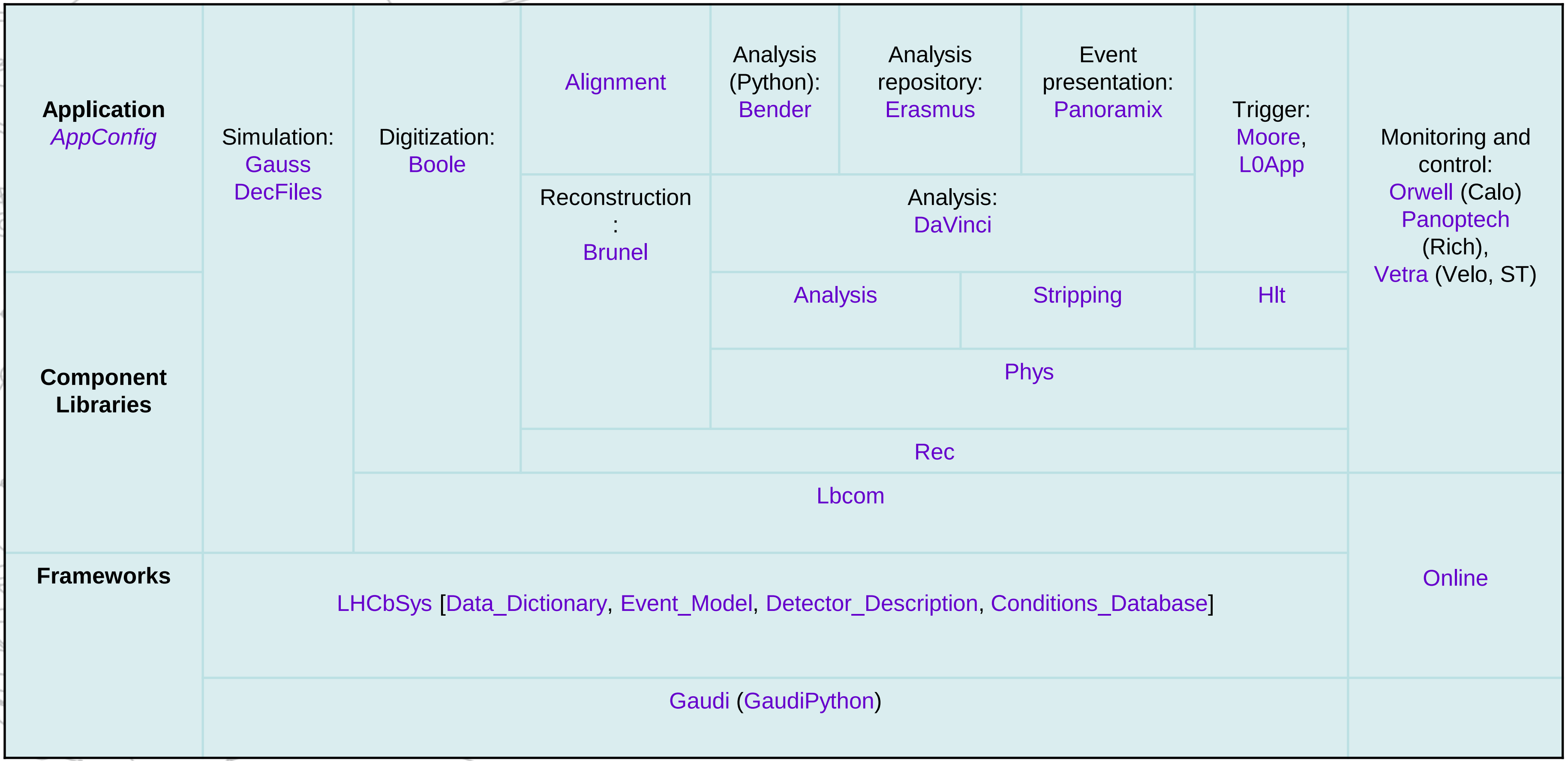
OFFLINE:

Geant
The best Geant ever

ROOT
Data Analysis Framework

POOL

Software for LHCb



Application *AppConfig*
Component Libraries
Frameworks
Simulation: Gauss DecFiles
Digitization: Boole
Alignment
Reconstruction : Brunel
Analysis (Python): Bender
Analysis repository: Erasmus
Event presentation: Panoramix
Trigger: Moore, L0App
Monitoring and control: Orwell (Calo) Panoptech (Rich), Vetra (Velo, ST)
Analysis: DaVinci
Analysis
Stripping
Hlt
Phys
Rec
Lbcom
Online
LHCbSys [Data_Dictionary, Event_Model, Detector_Description, Conditions_Database]
Gaudi (GaudiPython)

LHCb Schedule

2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022
LHC test	R1 Phase				LS1 Phase		R2 Phase			LS2 Phase		R3 Phase		

	Collision energy	Bunch length	Bunch Luminosity	Event Frequency	Event Size	Generated data (limit)	Stored data
R1	8 TeV	50ns	4e32/(cm^2*s)	40MHz	100KB	4TB/s	40MB/s
R2	13 TeV	25ns	4e32/(cm^2*s)	40MHz	100KB	4TB/s	2GB/s
R3	14 TeV	25ns	2e33/(cm^2*s)	30MHz	> 100KB	4TB/s	> 2GB/s

LS1 + R2:

- Simplifications in L0 i L1

LS2 + R3:

- HLT moved from cavern to surface
- Complete elimination of L0 i L1 (**Full Software Trigger**)

CERN software

- Multiple „big” frameworks
- Code developed by physicists
- Code developed in a hurry
- Detector systems specific knowledge
- Development criteria change over time
- High robustness & efficiency requirements

Manycore architectures

- Time and energy **costs**?
- **Programmability**?
- Deployment model and **scalability?**
- **Performance** tuning methodology?
- Future of MIC?

VS.

Work in progress

Activity	Status	Measurables
VCL library port for KNC	Completed?	- Good cooperation with VCL author Agner Fog (TUD, Copenhagen) - Code published in public domain (GPL) - Measurements gathered, - Article in review
LLVM as large code optimization platform	Hardware manufacturers need to put effort	- Possible methodology for both industry and academia
HEP benchmarking suite	Under development	- New benchmark suite and algorithm library for HEP available in public domain (permissive license)
Blog on Many-core	Continuous work	- cern.ch/manycore

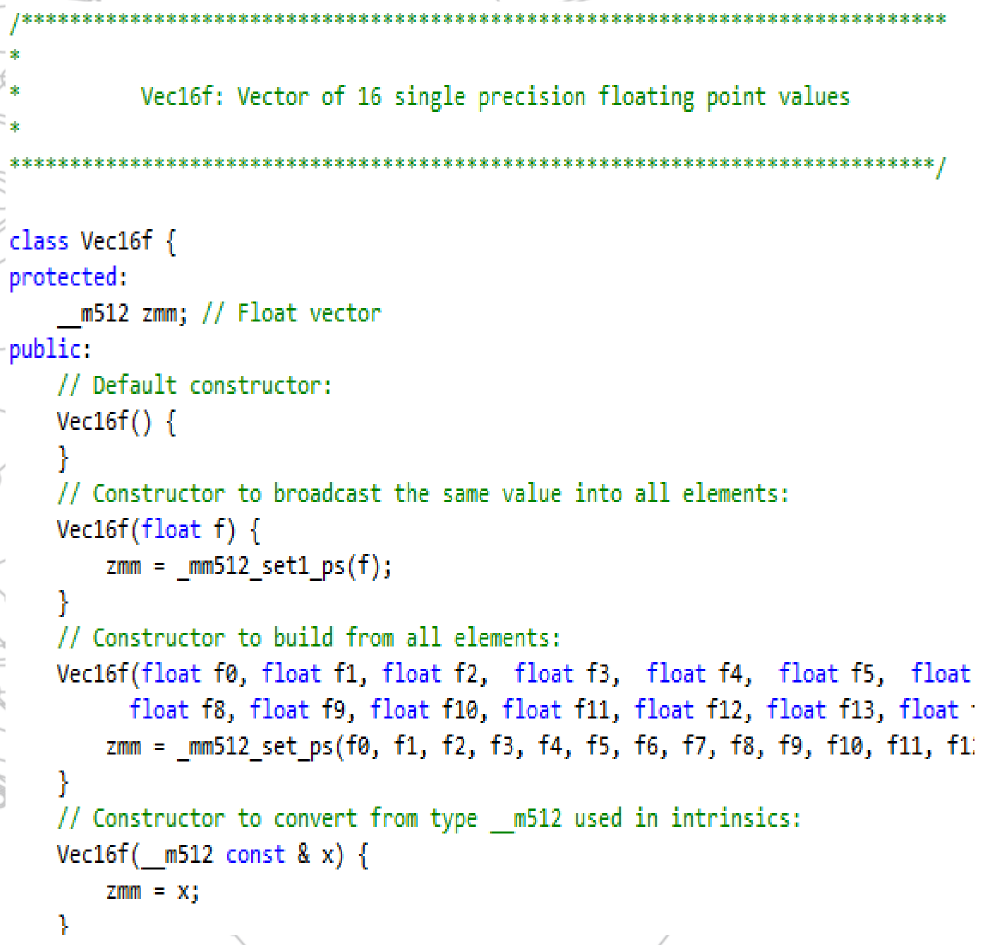
Wrong questions asked?

- *How do I measure performance?*
  - *Do you know what the metric is?*
- *How do I increase performance?*
  - *What are your hot-spots?*
- *How do I make my solution scalable?*
  - *What is your definition for scaling?*

Better questions?

- *How do I increase performance with minimal effort?*
  - *#pragma ...*
  - *Compile with –O3 –fastmath*
  - *Use faster library*
- *How do I choose proper metric?*
  - *Measure throughput*
  - *Measure latency*
  - *Measure memory utilization*
- *How do I create specification of my software?*
  - *Code IS the specification*

VCL and VCLKNC

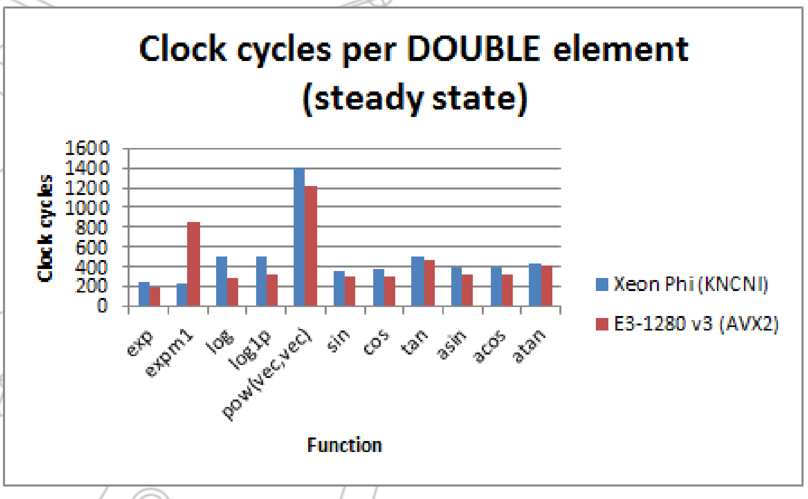


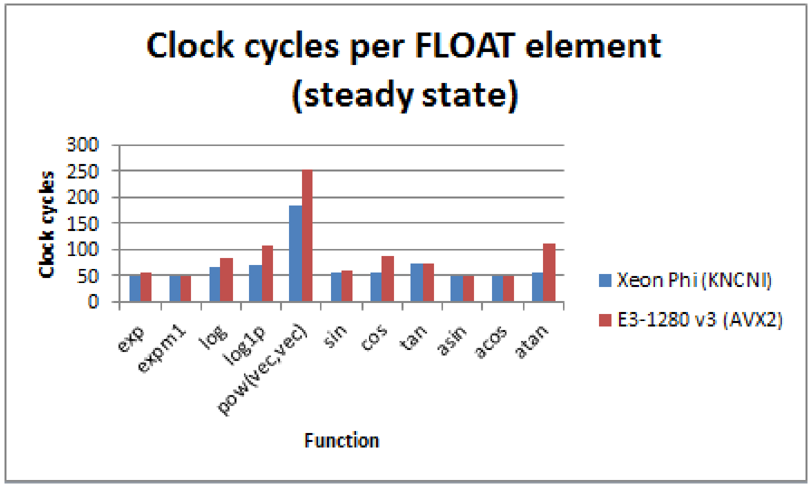
```
/*****************************************************************************
*
*          Vec16f: Vector of 16 single precision floating point values
*
*****************************************************************************/

class Vec16f {
protected:
    __m512 zmm; // Float vector
public:
    // Default constructor:
    Vec16f() {
    }
    // Constructor to broadcast the same value into all elements:
    Vec16f(float f) {
        zmm = _mm512_set1_ps(f);
    }
    // Constructor to build from all elements:
    Vec16f(float f0, float f1, float f2,  float f3,  float f4,  float f5,  float
          float f8, float f9, float f10, float f11, float f12, float f13, float
        zmm = _mm512_set_ps(f0, f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f1
    }
    // Constructor to convert from type __m512 used in intrinsics:
    Vec16f(__m512 const & x) {
        zmm = x;
    }
```

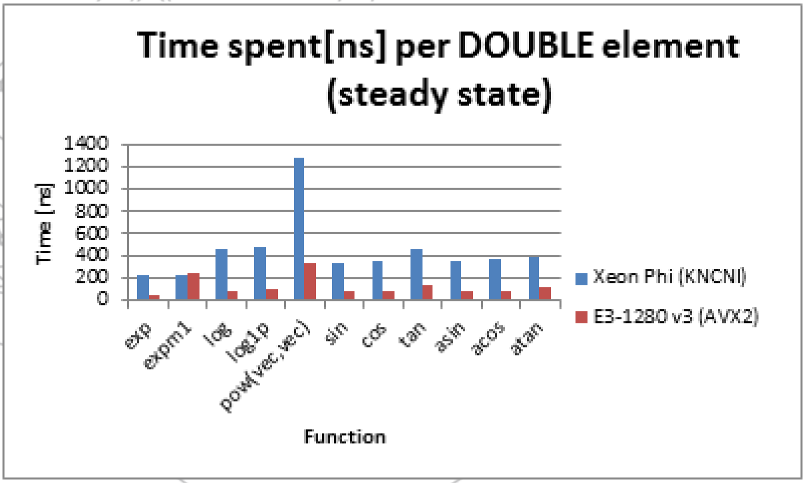
- SIMD vector abstraction layer
- Based on VCL library by Agner Fog (TUD, Copenhagen) www.agner.org

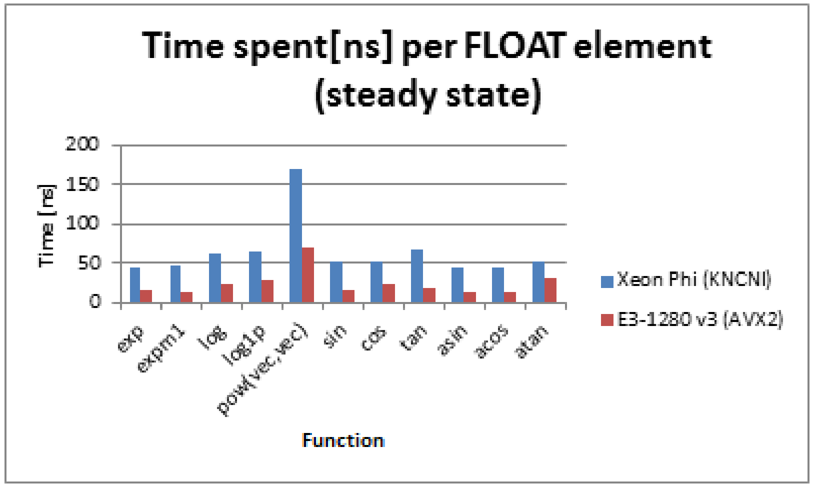
  Invaluable learning materials!
- Classes hiding SSE,AVXx
- VCLKNC – extension for IMCI (KNC) https://bitbucket.org/veclibknc/vclknc
- GPL, proprietary licensing possible

VCL: KNC vs XEON

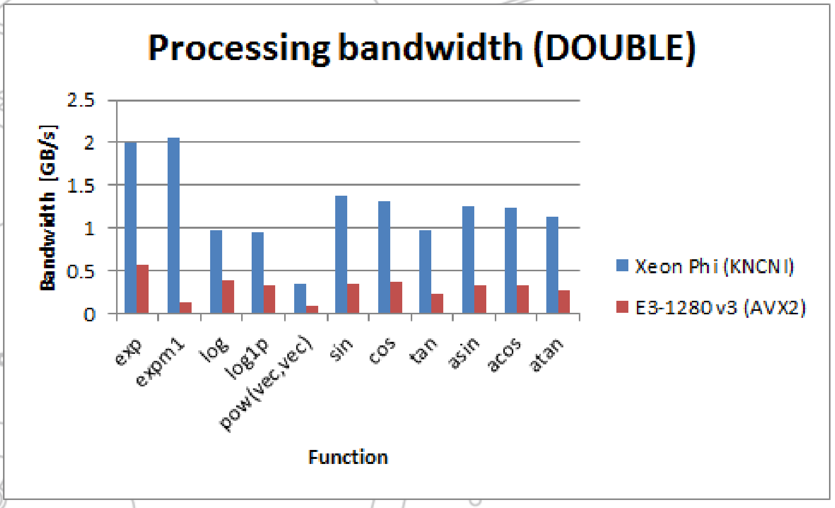


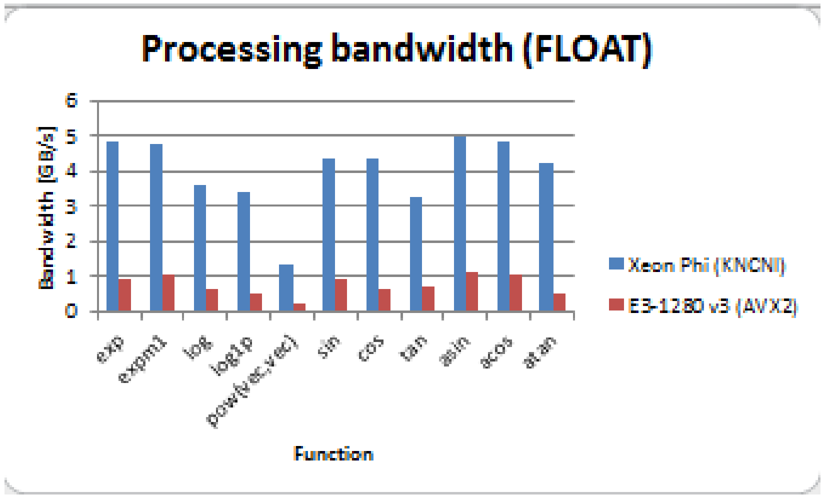






VCL: KNC vs XEON

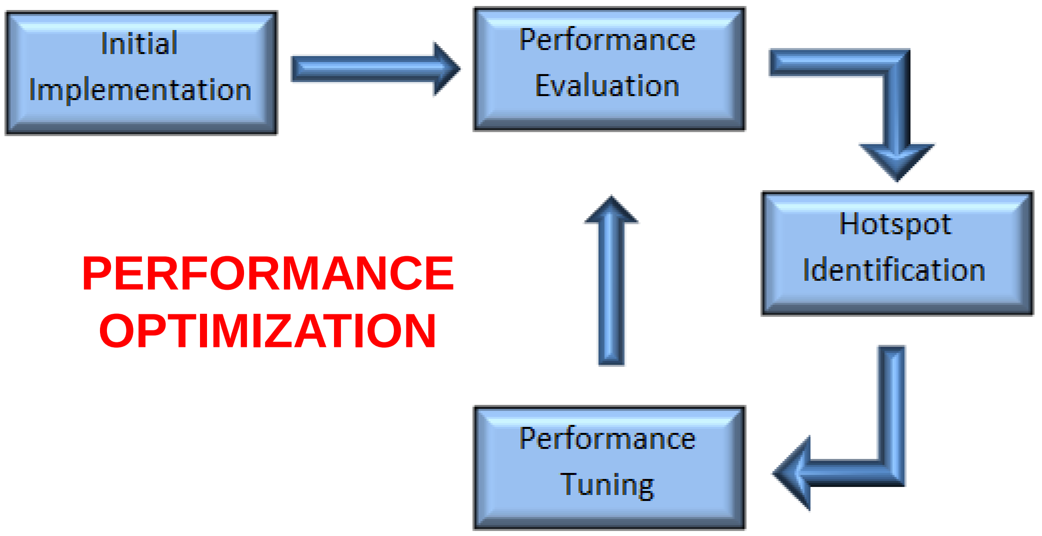




Conclusions:

- Explicit vectorization makes vectorization straightforward
- Intrinsics are not that complicated (but tricky sometimes)
- KNC core microarchitecture is not that bad
- Can we get higher frequency?
- Use floats instead of doubles!
- Throughput is promising.

Evolution vs. Revolution
(optimization vs. re-design)



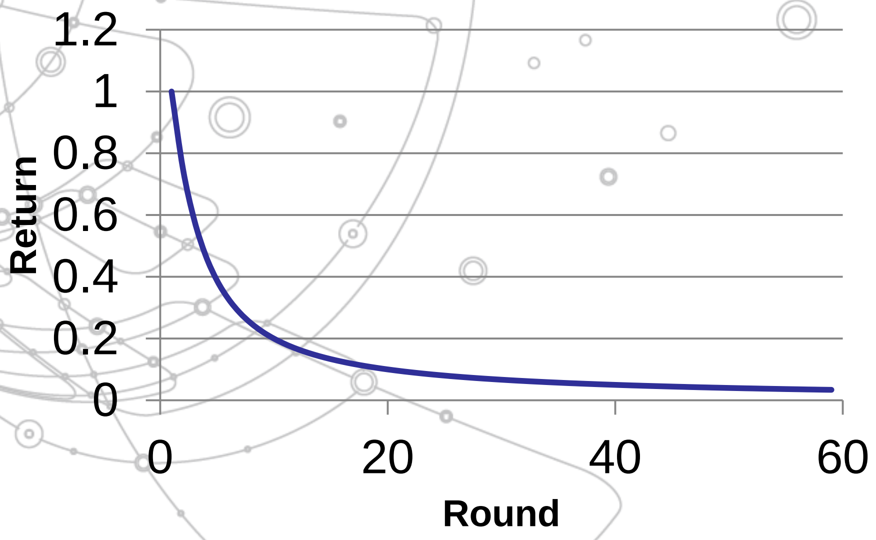
PERFORMANCE OPTIMIZATION

Law Of Diminishing Returns

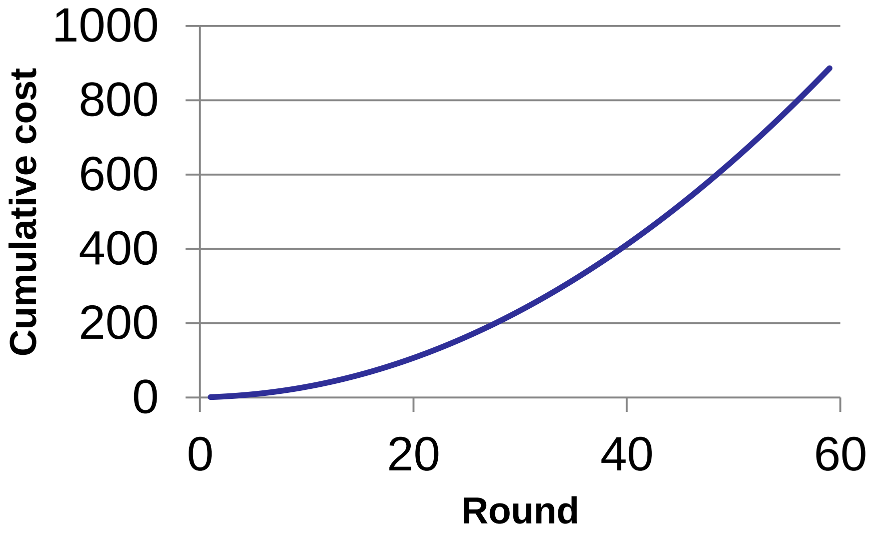
Suppose, for example, that **1 kilogram** of seed applied to a certain plot of land produces **one ton** of crop, that **2 kg of seed produces 1.5 tons**, and that **3 kg of seed produces 1.75 tons**.

Return (i-th round): $\frac{1}{N}\sum_{i=0}^{N}\frac{1}{2^{i-1}}$

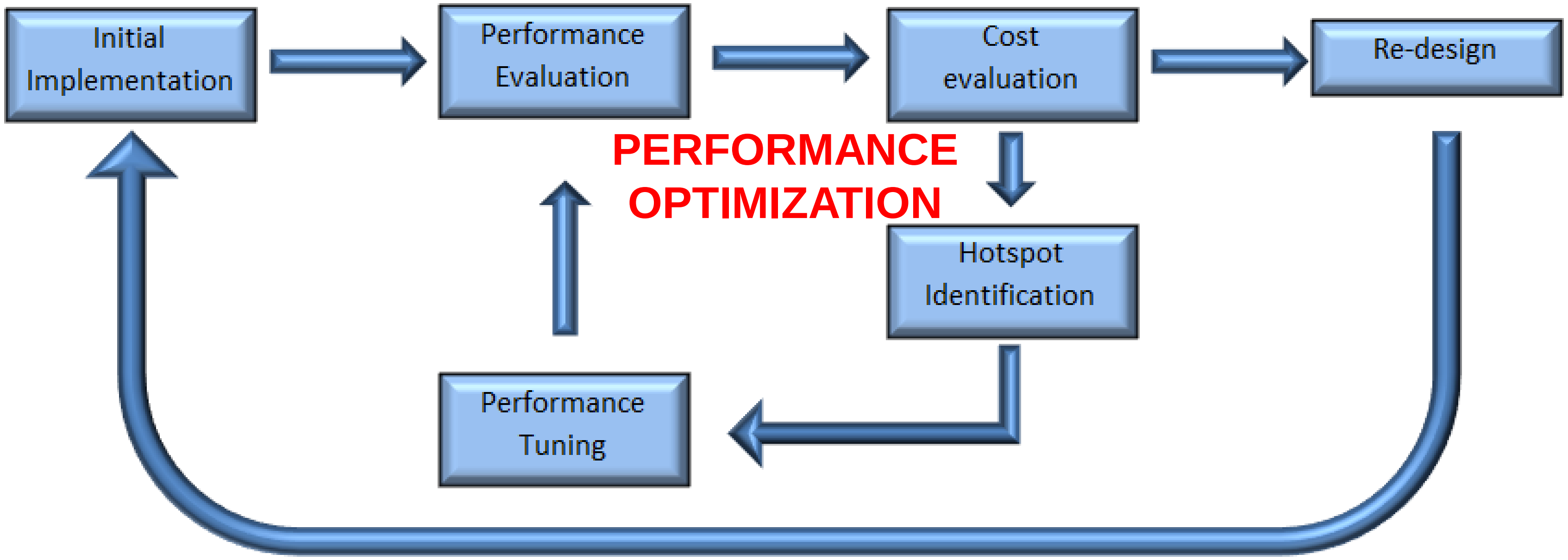
Diminishing returns...



...growing cost



Evolution vs. Revolution
(optimization vs. re-design)

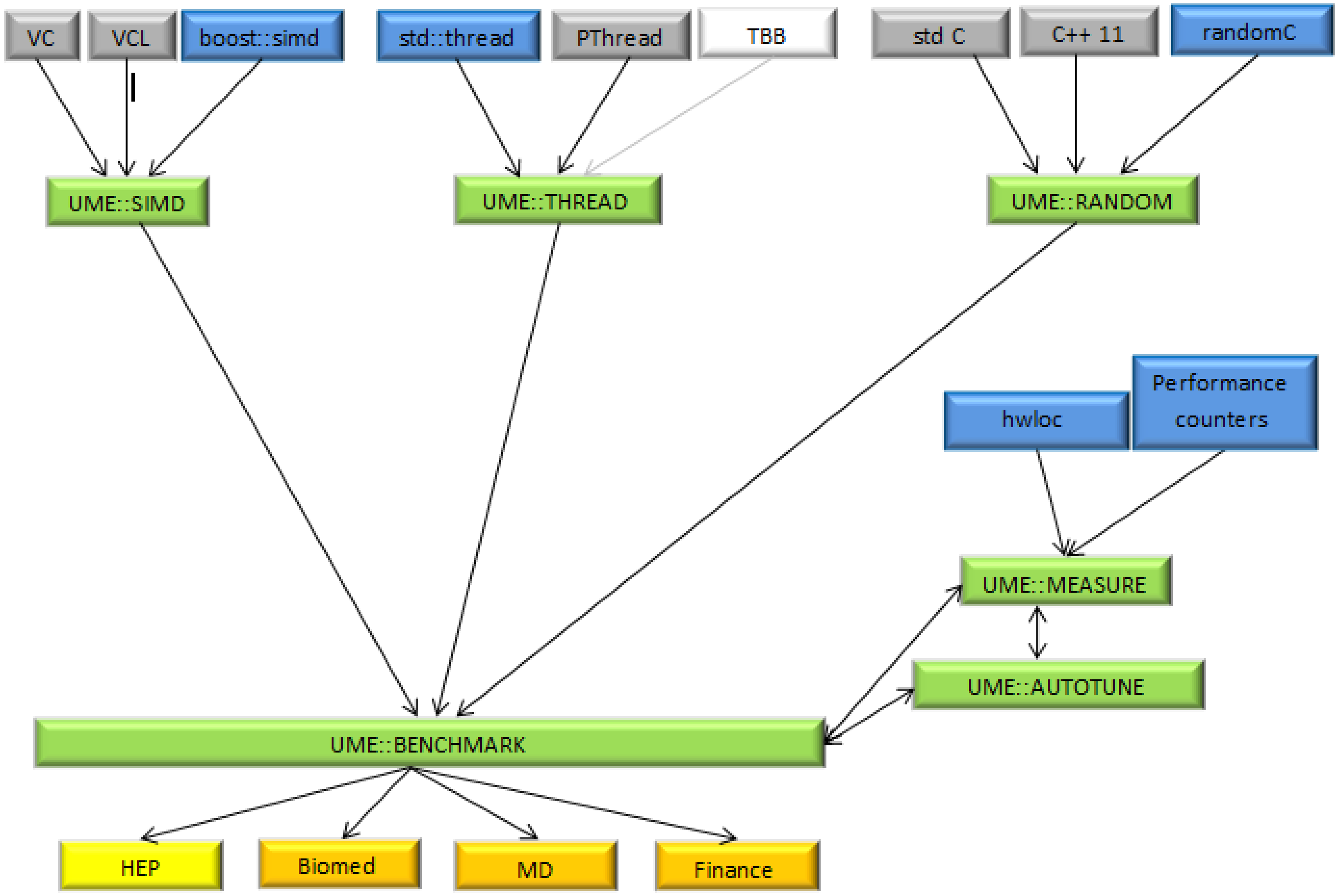


PERFORMANCE REVOLUTION

We need systematic revolution

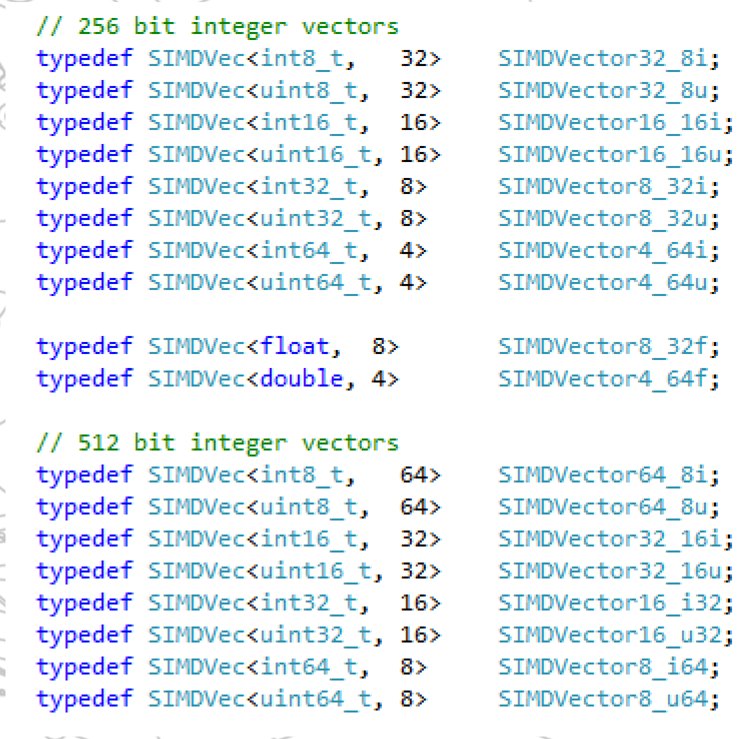
1) Write simplest code that solves your problem
   - We ALWAYS underestimate complexity!
   - Not sure what is proper SPECIFICATION before writing code down
   - Early optimization is overkill
2) Evaluate the cost of optimization and cost of redesign
   - Cost metric depends on project requirements!
   - You already know cost of initial implementation
3) Identify „hot spots” and optimize them
   - Hot spot is not only a function: it can be algorithm or structure
4) Repeat 2) until it is REASONABLE!
5) Write version 2 and start from the beginning
   - Don’t be afraid to do that! Now you have knowledge you didn’t have at stage 1)
   - Some components can and should be re-used

UME: basic structure



UME – Unified Multi/Manycore Environment

SIMD abstraction layer:



```
// 256 bit integer vectors
typedef SIMDVec<int8_t,   32>    SIMDVector32_8i;
typedef SIMDVec<uint8_t,  32>    SIMDVector32_8u;
typedef SIMDVec<int16_t,  16>    SIMDVector16_16i;
typedef SIMDVec<uint16_t, 16>    SIMDVector16_16u;
typedef SIMDVec<int32_t,  8>     SIMDVector8_32i;
typedef SIMDVec<uint32_t, 8>     SIMDVector8_32u;
typedef SIMDVec<int64_t,  4>     SIMDVector4_64i;
typedef SIMDVec<uint64_t, 4>     SIMDVector4_64u;

typedef SIMDVec<float,  8>       SIMDVector8_32f;
typedef SIMDVec<double, 4>       SIMDVector4_64f;

// 512 bit integer vectors
typedef SIMDVec<int8_t,   64>    SIMDVector64_8i;
typedef SIMDVec<uint8_t,  64>    SIMDVector64_8u;
typedef SIMDVec<int16_t,  32>    SIMDVector32_16i;
typedef SIMDVec<uint16_t, 32>    SIMDVector32_16u;
typedef SIMDVec<int32_t,  16>    SIMDVector16_i32;
typedef SIMDVec<uint32_t, 16>    SIMDVector16_u32;
typedef SIMDVec<int64_t,  8>     SIMDVector8_i64;
typedef SIMDVec<uint64_t, 8>     SIMDVector8_u64;
```

- VCL, VC, Boost::SIMD
- Library selection at compile time
- Uniform interface chosen after analysis of libraries
- Vector symetry problems resolved by emulation
- Possible to „plug-in” other libraries

Unified Multi/Manycore Environment (UME)

Next steps:

- „Other” abstraction layers
- Integrated benchmarking capabilities
  - Performance evaluation & cost evaluation
- Microbenchmarking platform characteristics
  - Canonical models of microarchitectures
  - Before or even during application compilation
- Canonical design of HEP algorithms
  - Ability to select parameters of the algorithm based on the platform specifics
  - Ability to re-use the algorithm for other applications (e.g.: Hough Transform, Kalman Filter)
  - Canonical algorithm FORCES data structures layout!!!
- Autotuning based on runtime information
  - It’s difficult to do „real” autotuning, we can gather runtime data and re-compile

UME: Three runs

Static identification:

- Identify hardware parameters:
  - Memory/core hierarchy
  - Memory and cross-core latencies
  - Single core performance
- Dump config file and recompile

Dynamic identification:

- Run domain specific microbenchmarks
- Select final software configuration:
- Dump final config file

Compile application for optimal set of SW components

Optimization methodology

Step 1: Write your algorithms using UME

- No need to know about underlying hardware
- Don’t worry about OS specific stuff
- Focus on performance

Step 2: Tune your software

- Static tuning allows selection of best libraries and some of algorithm parameters (compile time information)
- Dynamic tuning allows tuning for domain and specific data (runtime information

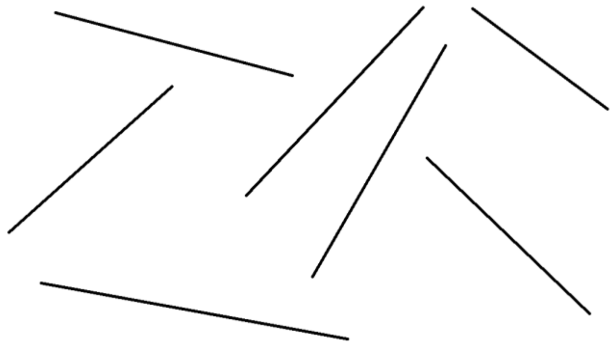
Step 3: Identify hot spots and specialize your algorithm

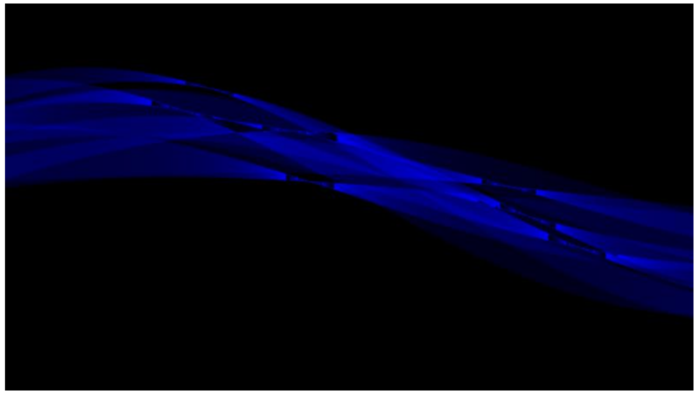
- Some tools for performance assesment integrated
- Specialize for HW/OS data intensity

HEP benchmarks

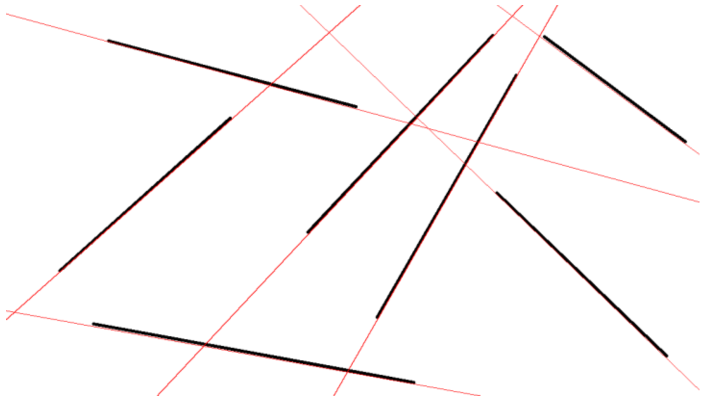
Hough transform (line detection):

$$r = x cos(\theta) + y\sin(\theta)$$



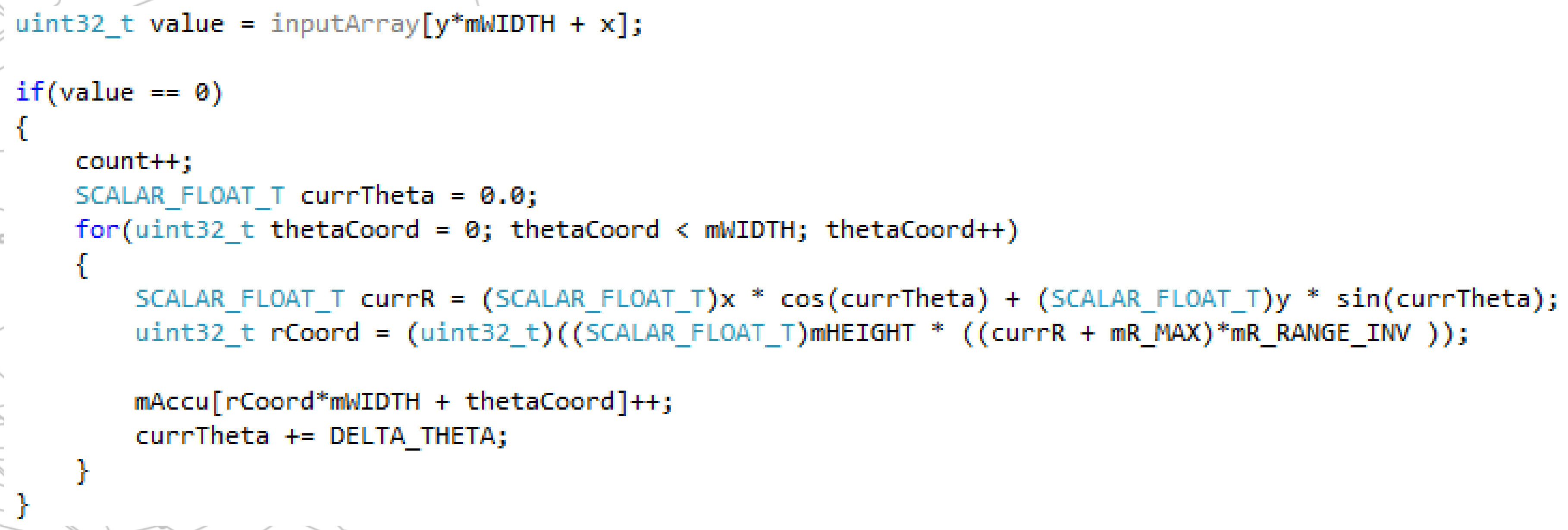


$$y = -\frac{\cos(\theta)}{\sin(\theta)}x + \frac{r}{\sin(\theta)}$$



Scalar vs. SIMD

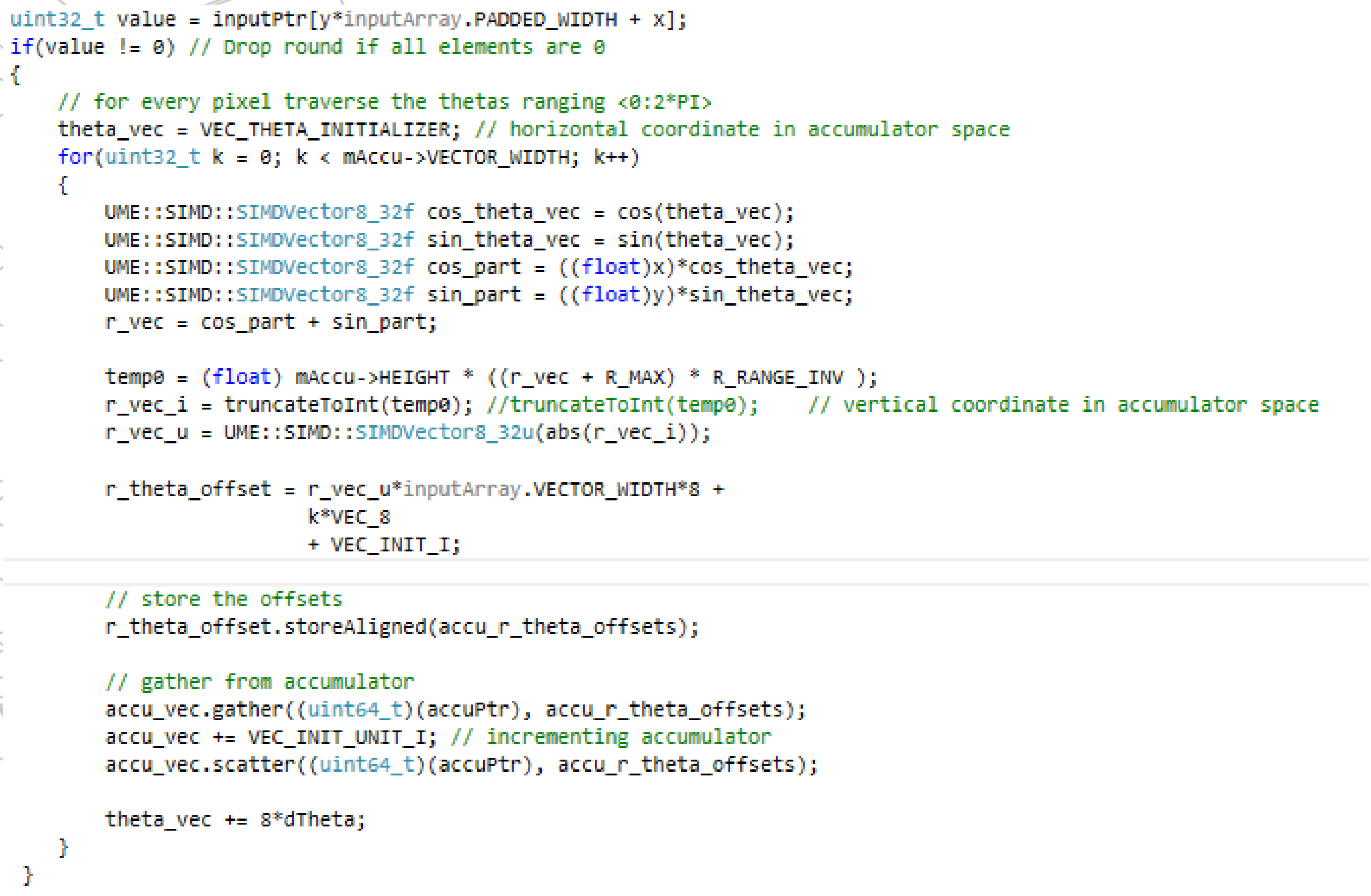
Scalar version:



```
uint32_t value = inputArray[y*mWIDTH + x];

if(value == 0)
{
    count++;
    SCALAR_FLOAT_T currTheta = 0.0;
    for(uint32_t thetaCoord = 0; thetaCoord < mWIDTH; thetaCoord++)
    {
        SCALAR_FLOAT_T currR = (SCALAR_FLOAT_T)x * cos(currTheta) + (SCALAR_FLOAT_T)y * sin(currTheta);
        uint32_t rCoord = (uint32_t)((SCALAR_FLOAT_T)mHEIGHT * ((currR + mR_MAX)*mR_RANGE_INV ));

        mAccu[rCoord*mWIDTH + thetaCoord]++;
        currTheta += DELTA_THETA;
    }
}
```

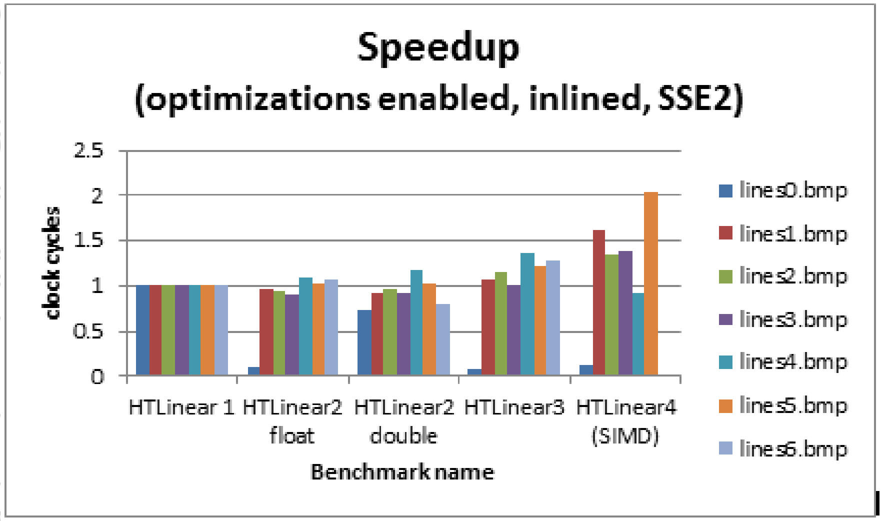
Scalar vs. SIMD

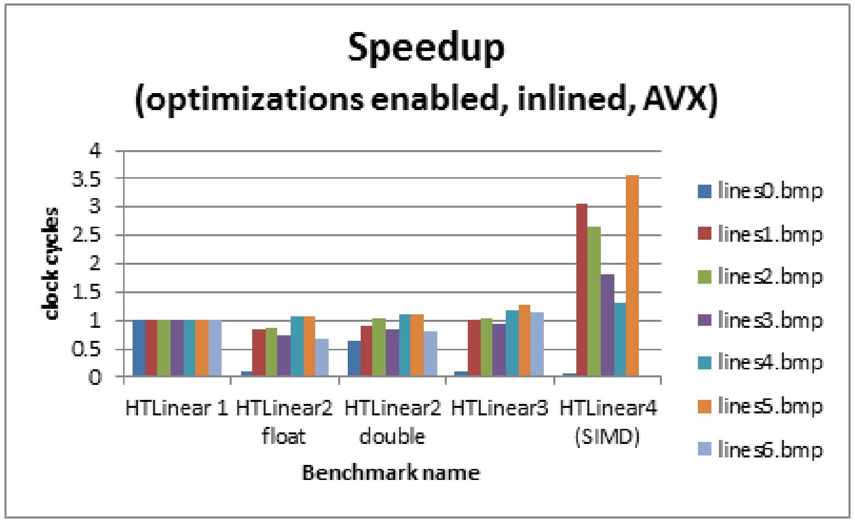
SIMD version:

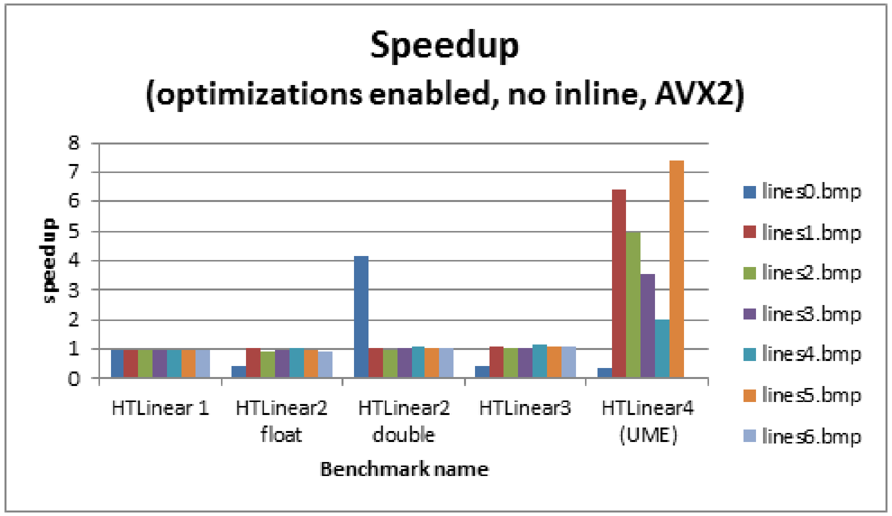


```
uint32_t value = inputPtr[y*inputArray.PADDED_WIDTH + x];
if(value != 0) // Drop round if all elements are 0
{
    // for every pixel traverse the thetas ranging <0:2*PI>
    theta_vec = VEC_THETA_INITIALIZER; // horizontal coordinate in accumulator space
    for(uint32_t k = 0; k < mAccu->VECTOR_WIDTH; k++)
    {
        UME::SIMD::SIMDVector8_32f cos_theta_vec = cos(theta_vec);
        UME::SIMD::SIMDVector8_32f sin_theta_vec = sin(theta_vec);
        UME::SIMD::SIMDVector8_32f cos_part = ((float)x)*cos_theta_vec;
        UME::SIMD::SIMDVector8_32f sin_part = ((float)y)*sin_theta_vec;
        r_vec = cos_part + sin_part;

        temp0 = (float) mAccu->HEIGHT * ((r_vec + R_MAX) * R_RANGE_INV );
        r_vec_i = truncateToInt(temp0); //truncateToInt(temp0);    // vertical coordinate in accumulator space
        r_vec_u = UME::SIMD::SIMDVector8_32u(abs(r_vec_i));

        r_theta_offset = r_vec_u*inputArray.VECTOR_WIDTH*8 +
                         k*VEC_8
                         + VEC_INIT_I;

        // store the offsets
        r_theta_offset.storeAligned(accu_r_theta_offsets);

        // gather from accumulator
        accu_vec.gather((uint64_t)(accuPtr), accu_r_theta_offsets);
        accu_vec += VEC_INIT_UNIT_I; // incrementing accumulator
        accu_vec.scatter((uint64_t)(accuPtr), accu_r_theta_offsets);

        theta_vec += 8*dTheta;
    }
 }
```

HT benchmark results







Key notes:

- Benchmarks 1 to 3: purely scalar
- Benchmark 4: explicit SIMD (8x32f vectors used)
- Exactly the same benchmark code for SSE2, AVX and AVX2
- Exactly the same benchmark code regardless of libraries selection
- Actual speedup depends on input data

Q&A

?

Thank you for attention!