

Summary of Programming for Concurrency and Co-Processors

Jim Pivarski and Vincenzo Innocente

March 29, 2018



FPGAs as Compute Accelerators

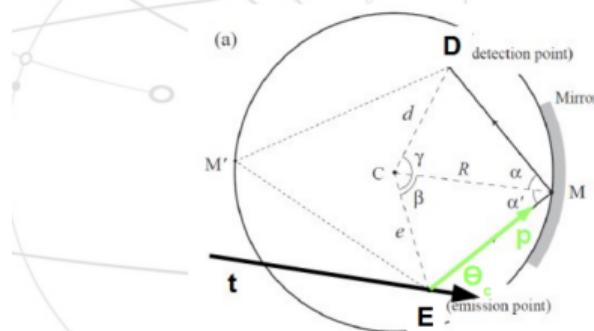
- Microsoft Catapult and Bing
 - Improve performance,
reduce power consumption
- Reduce the number of
von Neumann abstraction layers
 - Bit level operations
- Power only logic cells and registers needed
- Current test devices in LHCb
 - Nallatech PCIe with OpenCL
 - Intel® Xeon®+FPGA





Test Case: RICH PID Algorithm

- Calculate Cherenkov angle Θ_c for each track t and detection point D , not a typical FPGA algorithm
- RICH PID is not processed for every event, processing time is too long!



Reference: LHCb Note LHCb-98-040

Calculations:

- solve quartic equation
- cube root
- complex square root
- rotation matrix
- scalar/cross products





Compare Verilog - OpenCL



- Development time

2.5 months – 2 weeks

3400 lines Verilog – 250 lines C



Faster

- Performance

Cube root : x35 – x30

RICH : x35 – x26

Easier

Comparable performance

- FPGA resource usage Stratix® V

RICH Kernel	Verilog RTL	OpenCL
FPGA Resource Type	FPGA Resources used [%]	FPGA Resources used [%]
ALMs	88	63
DSPs	67	82
Registers	48	24

Similar resource usage



Andrei Gheata: VecCore: Expressing HEP algos in explicit SIMD types

How do we get it?

- Auto-vectorization
 - Compiler optimization converting repetitive scalar instructions (loops) to SIMD code
- Compiler pragmas
 - Code annotations persuading the compiler into vectorizing
 - OpenMP, CilkPlus
 - May not preserve exact scalar behavior
- SIMD libraries
 - VCL, Vc, UME::SIMD, [VecCore](#)
 - Explicit programming using specific vector types and operations
- Compiler intrinsics
 - Built-in inline compiler functions accessing architecture-specific vector instructions
- Assembly
 - The really low-level stuff on top of HW implementation

```
float a[N], b[N], c[N];  
  
for (int i = 0; i < N; i++)  
    a[i] = b[i] * c[i];
```

```
float a[N], b[N], c[N];  
  
#pragma omp simd  
#pragma ivdep  
for (int i = 0; i < N; i++)  
    a[i] = b[i] * c[i];
```

```
#include <VecCore/VecCore>  
using Float_v =  
    backend::VcVector::Float_v;  
Float_v a = b * c;
```

```
#include <x86intrin.h>  
__m256 a, b, c;  
  
a = _mm256_mul_ps(b, c);  
  
asm volatile("vmulps %ymm1,  
%ymm0");
```

Programmability
Performance
Portability

Andrei Gheata: VecCore: Expressing HEP algos in explicit SIMD types

VecCore operations

G. Amadio, ACAT'17

```
namespace vecCore {

    template <typename T> struct TypeTraits;
    template <typename T> using Mask = typename TypeTraits<T>::MaskType;
    template <typename T> using Index = typename TypeTraits<T>::IndexType;
    template <typename T> using Scalar = typename TypeTraits<T>::ScalarType;

    // Vector Size
    template <typename T> constexpr size_t Vectorsize();

    // Get/Set
    template <typename T> Scalar<T> Get(const T &v, size_t i);
    template <typename T> void Set(T &v, size_t i, Scalar<T> const val);

    // Load/Store
    template <typename T> void Load(T &v, Scalar<T> const *ptr);
    template <typename T> void Store(T const &v, Scalar<T> *ptr);

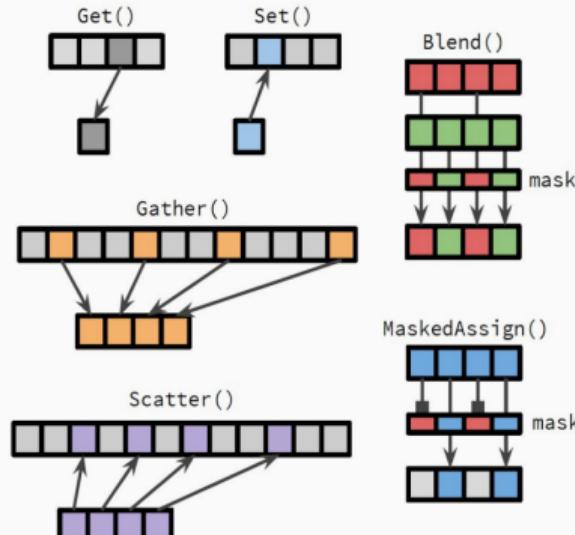
    // Gather/Scatter
    template <typename T, typename S = Scalar<T>>
    T Gather(S const *ptr, Index<T> const &idx);

    template <typename T, typename S = Scalar<T>>
    void Scatter(T const &v, S *ptr, Index<T> const &idx);

    // Masking/Blending
    template <typename M> bool MaskFull(M const &mask);
    template <typename M> bool MaskEmpty(M const &mask);

    template <typename T> void MaskedAssign(T &dst, const Mask<T> &mask, const T &src);
    template <typename T> T Blend(const Mask<T> &mask, const T &src1, const T &src2);

} // namespace vecCore
```

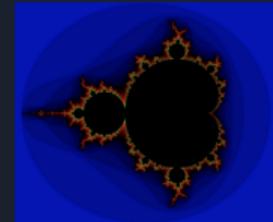


[https://github.com/root-project/veccore/
blob/master/doc/API.md](https://github.com/root-project/veccore/blob/master/doc/API.md)

Andrei Gheata: VecCore: Expressing HEP algos in explicit SIMD types

Internal vectorization example: the Mandelbrot set

Iterate $f(z) = z^2 + c$ N times and check if z diverges

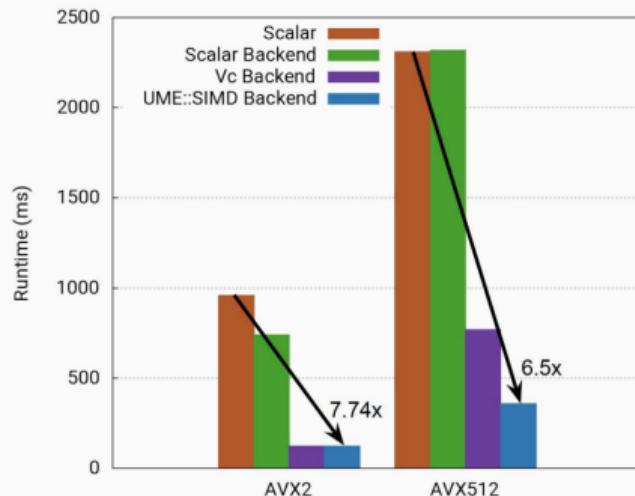


scalar

```
template<typename T>
void mandelbrot(T xmin, T xmax,
                T ymin, T ymax,
                size_t max_iter,
{
    T dx = (xmax - xmin) / T(nx);
    T dy = (ymax - ymin) / T(ny);
    for (size_t i = 0; i < nx; ++i) {
        for (size_t j = 0; j < ny; ++j) {
            size_t k = 0;
            T x = xmin + T(i) * dx, cr =
            T y = ymin + T(j) * dy, ci =
            do {
                x = zr*zr - zi*zi + cr;
                y = 2.0 * zr*zi + ci;
                zr = x;
                zi = y;
            } while (++k < max_iter &
image[ny*i+j] = k;
        }
    }
}
```

VecCore

Performance with AVX2 and AVX512 using different backends



```
T> xmax, size_t nx,
T> ymax, size_t ny,
iter, unsigned char *image)
```

```
Set<T>(iota, i, i);
```

```
* dy;
```

```
<T>()) {
```

```
z = y;
```

```
i, ++k);
```

```
pty(m));
```

```
+k)
```

```
set(kv, k);
```

Martin Durant: Dask: distributed computing for scientific Python

DASK: HOW TO SCALE UP WITH A MINIMUM OF HASSLE

Run dask on your laptop, or on a large cluster: just specify the scheduler address.

In [1]:

```
import dask.distributed
client = dask.distributed.Client('dask-scheduler:8786')
```

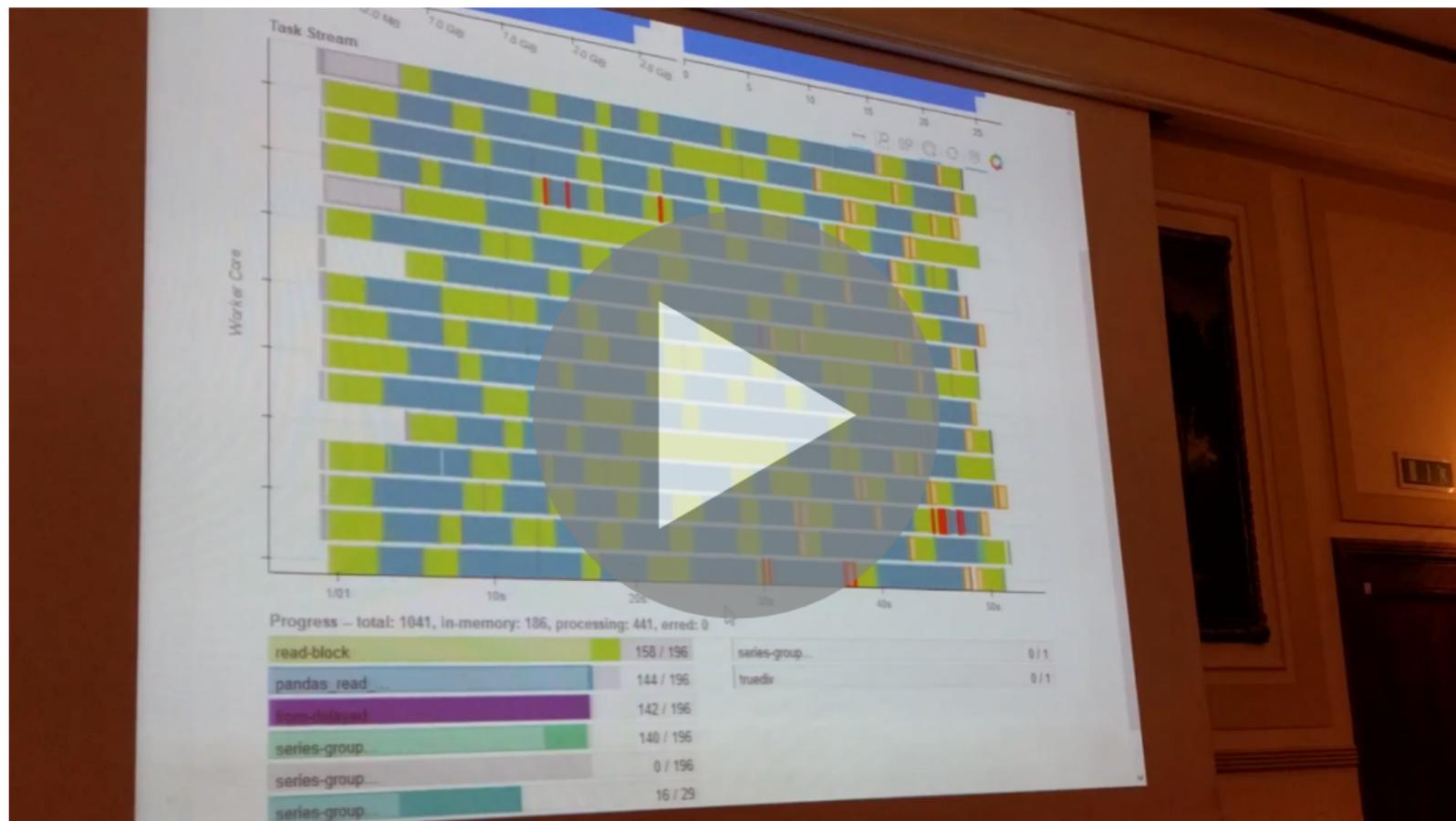
Out[1]:

CLIENT	CLUSTER
• Scheduler: http://dask-scheduler:8786	• Workers: 8
• Dashboard: http://dask-scheduler:8787/metrics	• Cores: 16
	• Memory: 54.19 GB

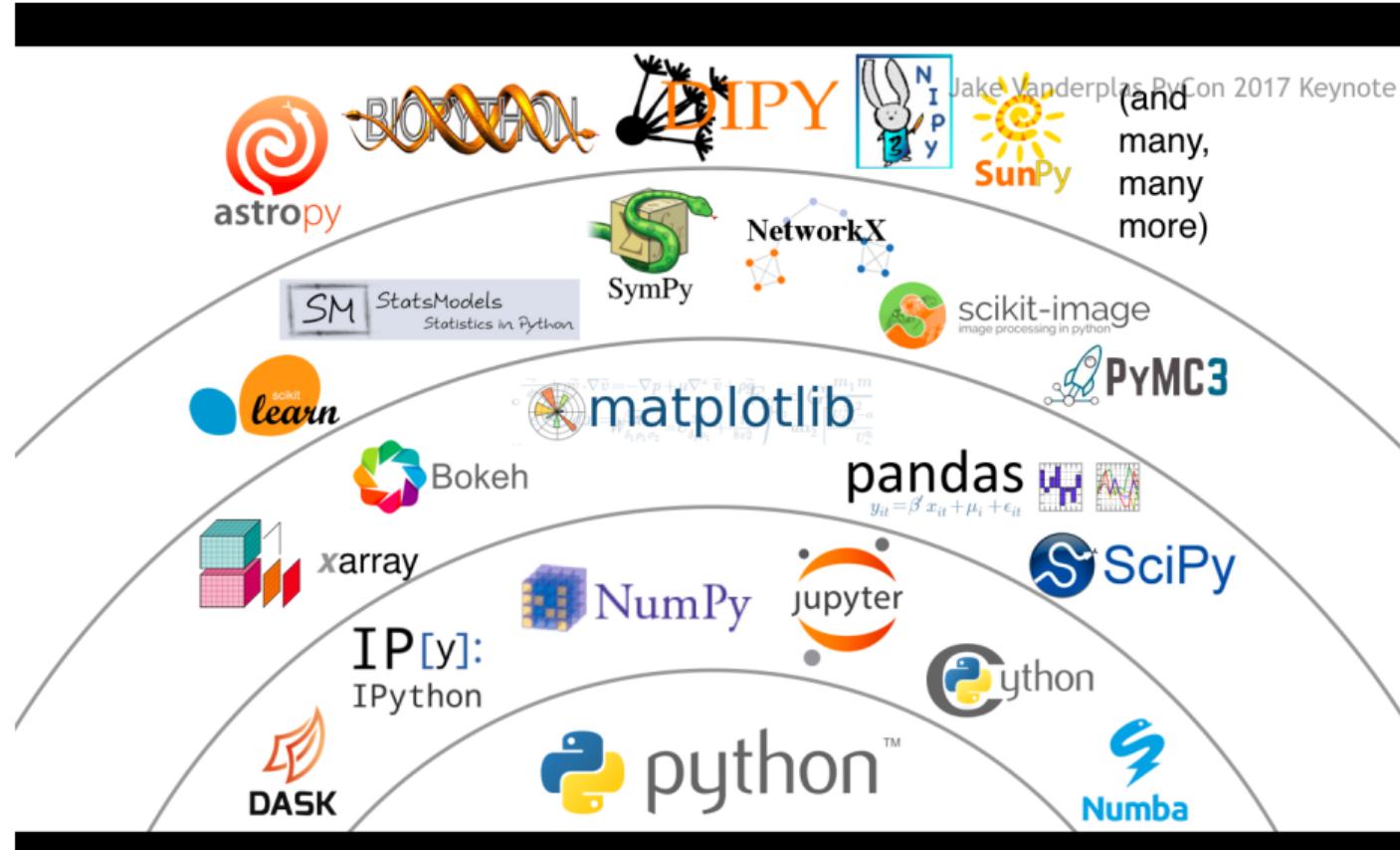
```
@dask.delayed
def f(x, y):
    do_thing_with_inputs
    return output
```

WLCG/HSF Workshop 2018

Martin Durant: Dask: distributed computing for scientific Python



Martin Durant: Dask: distributed computing for scientific Python



Challenges for massive parallelism

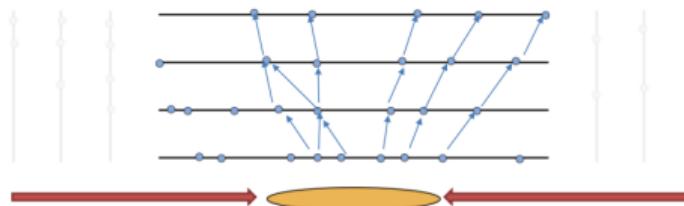


- We have small matrices (5×5 / 6×6)
 - Typical approach for matrix operation vectorization does not scale for us
 - * = vectorize the calculations of a single matrix operation
- Pattern recognition problem is combinatorial with dynamic output size
 - Need to create new track candidates on the fly
 - In general need dynamic memory allocations
- Traditional algorithms have lots of control logic
 - Harmful for SIMD
- Data transfers from CPU/host memory to accelerator(s)
 - For short computations data transfer time may dominate
- E.g. in Intel Skylake the use of AVX/AVX-512 instructions slows down the frequency of the core
 - So to benefit from AVX, significant portion of computational kernel has to be vectorized

Matti Kortelainen: Parallelized tracking algorithms



- First create doublets from hits of pairs
- Take a third layer and propagate only the generated doublets
- Consider a fourth layer and propagate triplets
- Store found quadruplets and start from another pair of layers

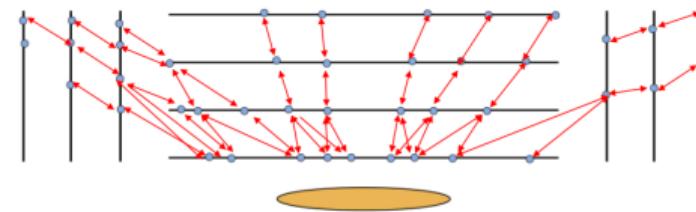


14

Matti Kortelainen: Parallelized tracking algorithms



- The CA is a track seeding algorithm designed for parallel architectures
- It requires a list of layers and their pairings
 - A graph of all the possible connections between layers is created
 - Doublets aka Cells are created for each pair of layers (compatible with a region hypothesis)
 - Fast computation of the compatibility between two connected cells
 - No knowledge of the world outside adjacent neighboring cells required, making it easy to parallelize
- However this is not a static problem, not at all...



Now a library, v0.1
<https://github.com/HEP-SF/TrickTrack>

17